

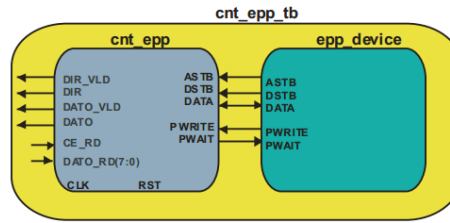
Modelado y síntesis de sistemas electrónicos digitales

PRÁCTICA LABORATORIO

CHRISTOPHER HYDE PEINADO 09046047D

1. CNT_EPP

1.1.1.- Código VHDL de la entidad *cnt_epp* y del banco de pruebas utilizado en su simulación. También se debe proporcionar el código de la entidad *epp_device*.



A) Código VHDL de la entidad *cnt_epp*:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity cnt_epp is
  port (
    CLK      : in  std_logic;
    RST      : in  std_logic;
    ASTRB    : in  std_logic;
    DSTRB    : in  std_logic;
    DATA    : inout std_logic_vector(7 downto 0);
    PWRITE   : in  std_logic;
    PWAIT    : out std_logic;
    DATO_RD  : in  std_logic_vector(7 downto 0);
    CE_RD    : out std_logic;
    DIR      : out std_logic_vector(7 downto 0);
    DIR_VLD  : out std_logic;
    DATO     : out std_logic_vector(7 downto 0);
    DATO_VLD : out std_logic;
  );
end entity;

architecture rtl of cnt_epp is

  signal S1:std_logic;
  signal S11:std_logic;
  signal Q:std_logic;

  signal S2:std_logic;
  signal S22:std_logic;
  signal Q2:std_logic;

begin
  -----ADDRESS-----
  AddrBistableD1:process (CLK,RST)
  begin
    if (RST='1') then
      Q <= '0';
    elsif (CLK'event and CLK='1')then
      Q <= ASTRB;
    end if;
  end process;

  S1<= ASTRB and not Q;
  S11<=S1 and not PWRITE;

  AddrBistableD2:process (CLK,RST)
  begin
    if (RST='1') then
      DIR_VLD <= '0';
    elsif (CLK'event and CLK='1')then
      DIR_VLD <= S11;
    end if;
  end process;
```

```
AddrBistableD3:process (CLK,RST,S11)
begin
  if (RST='1') then
    DIR <= (others=>'0');
  elsif (CLK'event and CLK='1')then
    if (S11='1') then
      DIR <= DATA;
    end if;
  end if;
end process;

-----DATA-----
DataBistableD1:process (CLK,RST)
begin
  if (RST='1') then
    Q2 <= '0';
  elsif (CLK'event and CLK='1')then
    Q2 <= DSTRB;
  end if;
end process;

S2<= DSTRB and not Q2;
S22<=S2 and not PWRITE;

DataBistableD2:process (CLK,RST)
begin
  if (RST='1') then
    DATO_VLD <= '0';
  elsif (CLK'event and CLK='1')then
    DATO_VLD <= S22;
  end if;
end process;

DataBistableD3:process (CLK,RST,S22)
begin
  if (RST='1') then
    DATO <= (others=>'0');
  elsif (CLK'event and CLK='1')then
    if (S22='1') then
      DATO <= DATA;
    end if;
  end if;
end process;
```

```

--tristate buffer
DATA<= DATO_RD when ((PWRITE= '1') and (DSTRB='1')) else (others => 'Z');

PWAITbiestableD:process (CLK,RST,ASTRB,DSTRB,PWRITE)
begin
    if (RST='1') then
        PWAIT <= '0';
    elsif (CLK'event and CLK='1')then
        if ((PWRITE='0') and (ASTRB='0')) or ((PWRITE='0') and (DSTRB='0')) or ((PWRITE='1') and (DSTRB='0')) then
            PWAIT <= '1';
        else
            PWAIT <= '0';
        end if;
    end if;
end process;

CE_RDbiestableD:process (CLK,RST,PWRITE)
begin
    if (RST='1') then
        CE_RD <= '1';
    elsif (CLK'event and CLK='1')then
        if (PWRITE='1') then
            CE_RD <= DSTRB;
        end if;
    end if;
end process;

```

B) Código VHDL de la entidad **epp_device** modificado para realizar 3 escrituras y una lectura:

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;
use ieee.std_logic_textio.all;

entity epp_device is
    port (
        DATA : inout std_logic_vector(7 downto 0);
        PWRITE : out std_logic;
        DSTRB : out std_logic;
        ASTRB : out std_logic;
        PWAIT : in std_logic);
end epp_device;

architecture sim of epp_device is
    constant T_clk_epp : time := 100 ns; -- Internal clock period.
    signal clk_epp : std_logic := '0'; -- Internal clock signal.
    signal read_value : std_logic_vector(7 downto 0) := (others => '0');
    constant dir_freq : std_logic_vector( 7 downto 0) := x"F0";
    constant dir_dpram1 : std_logic_vector( 7 downto 0) := x"A1";
    constant dir_dpram2 : std_logic_vector( 7 downto 0) := x"A2";
    constant EPP_cycle_length: natural:= 10;

begin
    -- internal clock signal generation.
    clk_epp <= not(clk_epp) after T_clk_epp/2;

```

```

Process
procedure epp_cicle ( address : in std_logic_vector(7 downto 0);
                    data_io : inout std_logic_vector(7 downto 0);
                    r_w : in character) is
begin
    wait until clk_epp = '1';
    PWRITE <= '0';
    wait until clk_epp = '1';
    ASTRB <= '0';
    data <= address;
    wait for T_clk_epp*EPP_cycle_length;
    ASTRB <= '1';
    wait until clk_epp = '1';
    data <= (others => 'Z');
    PWRITE <= '1';
    wait until clk_epp = '1';
    wait for T_clk_epp*EPP_cycle_length;

    -----

    if r_w = 'w' then -- write cicle
        PWRITE <= '0';
        data <= data_io;
    end if;

    -----

    wait until clk_epp = '1';
    DSTRB <= '0';
    wait for T_clk_epp*EPP_cycle_length;

    -----

    if r_w = 'r' then -- read cicle
        data_io:= data;
    end if;

    -----

    DSTRB <= '1';
    wait until clk_epp = '1';
    data <= (others => 'Z');
    PWRITE <= '1';
    wait until clk_epp = '1';
end procedure;

```

```

file arch_in : text ;
variable bf : line;
  variable dato : std_logic_vector(7 downto 0);
  variable dir : std_logic_vector(7 downto 0);
begin
  --inicialización
  data <= (others => 'Z');
  PWRITE <= '1';
  DSTRB <= '1';
  ASTRB <= '1';
  dir := (others => '0');

  -----first write values-----
wait for 130 ns;
  DIR:=dir_dpram1;
  DATO:=X"34";
  epp_cycle ( address => dir,
              data_io => dato,
              r_w   => 'w');

  -----second write values-----
  wait for 130 ns;
  DIR:=dir_dpram2;
  DATO:=X"44";
  epp_cycle ( address => dir,
              data_io => dato,
              r_w   => 'w');

  -----third write values-----
  wait for 130 ns;
  DIR:=dir_freq;
  DATO:=X"54";
  epp_cycle ( address => dir,
              data_io => dato,
              r_w   => 'w');

  -----first read values-----
wait for 130 ns;
  DIR:=X"FF";
  epp_cycle ( address => dir,
              data_io => dato,
              r_w   => 'r');
  read_value<=dato;
wait for 1 us;

report "FIN CICLO R/W" severity failure;

  end process;
end sim;

```

C) Código VHDL del **testbench** de simulación **cnt_epp_tb**:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY cnt_epp_tb IS
END cnt_epp_tb;

ARCHITECTURE rtl OF cnt_epp_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT cnt_epp
    PORT(
        CLK : IN  std_logic;
        RST : IN  std_logic;
        ASTRB : IN  std_logic;
        DSTRB : IN  std_logic;
        DATA : INOUT std_logic_vector(7 downto 0);
        PWRITE : IN  std_logic;
        PWAIT : OUT  std_logic;
        DATO_RD : IN  std_logic_vector(7 downto 0);
        CE_RD : OUT  std_logic;
        DIR : OUT  std_logic_vector(7 downto 0);
        DIR_VLD : OUT  std_logic;
        DATO : OUT  std_logic_vector(7 downto 0);
        DATO_VLD : OUT  std_logic
    );
    END COMPONENT;

    COMPONENT epp_device
    port (
        DATA : inout std_logic_vector(7 downto 0);
        PWRITE : out  std_logic;
        DSTRB : out  std_logic;
        ASTRB : out  std_logic;
        PWAIT : in   std_logic;
    );
    END COMPONENT;

    signal ASTRB_comm:std_logic;
    signal DSTRB_comm:std_logic;
    signal PWRITE_comm:std_logic;
    signal PWAIT_comm:std_logic;
    signal DATA_comm:std_logic_vector(7 downto 0);

    --cnt_epp

    signal  DATO_RD_comm :std_logic_vector(7 downto 0):="AA";
    signal  CE_RD :std_logic;
    signal  DIR :std_logic_vector(7 downto 0);
    signal  DIR_VLD : std_logic;
    signal  DATO :std_logic_vector(7 downto 0);
    signal  DATO_VLD :std_logic;

    -- Clock period definitions
    constant CLK_period : time := 10 ns;
    signal clk : std_logic;
    signal rst : std_logic:='1';
```

```
BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: cnt_epp PORT MAP (
    CLK => CLK,
    RST => RST,
    ASTRB => ASTRB_comm,
    DSTRB => DSTRB_comm,
    DATA => DATA_comm,
    PWRITE => PWRITE_comm,
    PWAIT => PWAIT_comm,
    DATO_RD => DATO_RD_comm,
    CE_RD => CE_RD,
    DIR => DIR,
    DIR_VLD => DIR_VLD,
    DATO => DATO,
    DATO_VLD => DATO_VLD
);

    eppDevice:epp_device
    port map(
        DATA=>DATA_comm,
        PWRITE=>PWRITE_comm,
        DSTRB=>DSTRB_comm,
        ASTRB=>ASTRB_comm,
        PWAIT=>PWAIT_comm
    );

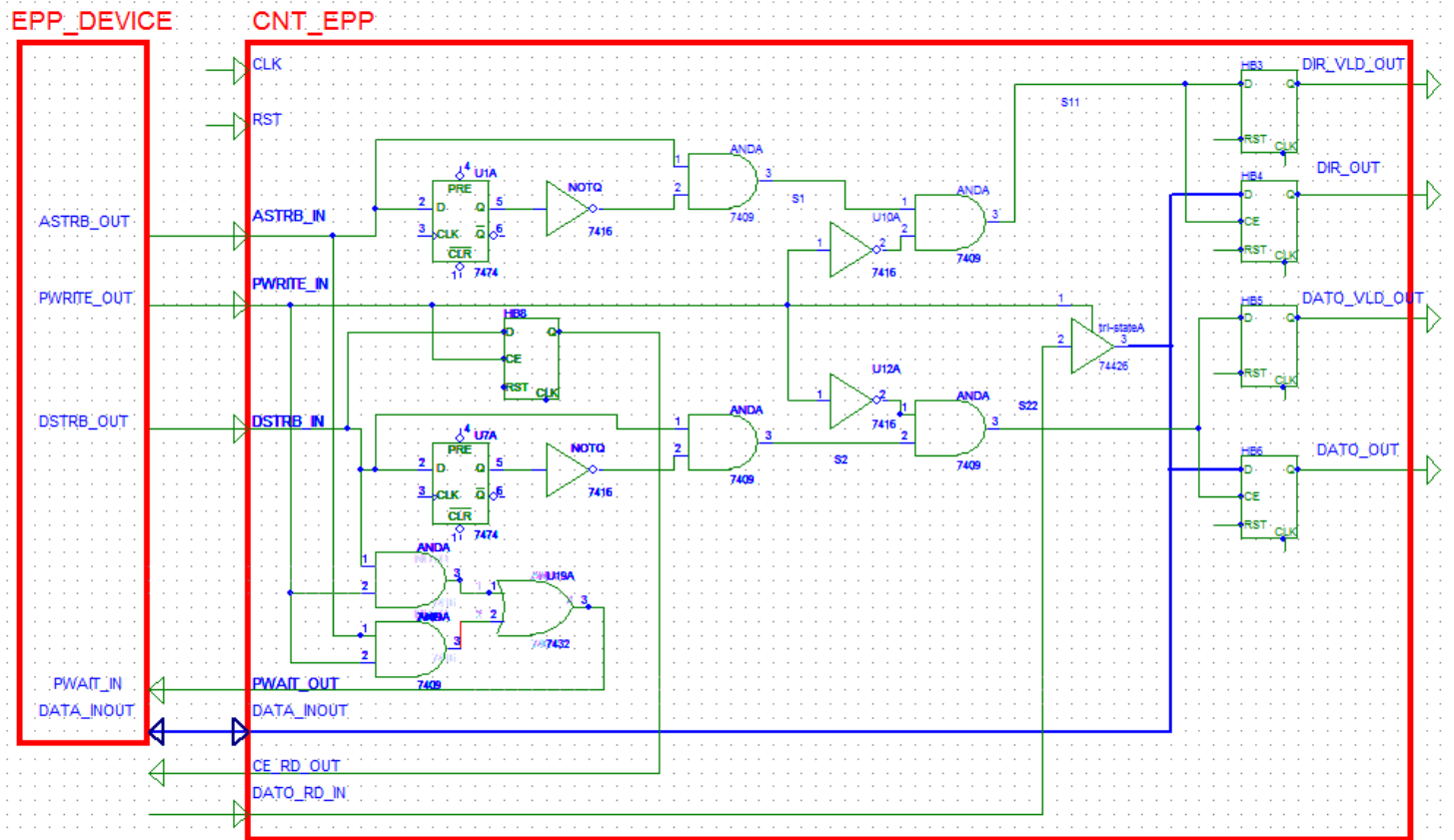
    -- Clock process definitions
    CLK_process :process
    begin
        CLK <= '0';
        wait for CLK_period/2;
        CLK <= '1';
        wait for CLK_period/2;
    end process;

    RST_process :process
    begin
        wait for 5 ns;
        RST <= '0';
    end process;

END rtl;
```

1.1.2- Razonamiento de por qué se ha adoptado la solución presentada.

El código presentado anteriormente es la implementación del siguiente esquema:



Gracias a este circuito se ha podido llegar a una solución óptima para el diseño del módulo

cnt_epp.

Éste módulo recibe los datos de temporización del **epp_device** y proporciona las salidas de Dir y Dato y Dir y Dato válidos, junto con señales de pwait y CE_rd.


El modulo está controlado por el reloj del sistema CLK, luego se trata de un módulo con sincronismo.

1.1.3.- Pantallazo de simulación donde se refleje que la entidad **cnt_esp** funciona correctamente. Para ello se realizarán tres ciclos de escritura, uno en cada uno de las direcciones que se van a utilizar en el diseño (*dir_frec*, *dir_dpram1* y *dir_dpram2*) y una operación de lectura en la dirección a elegir por el alumno.



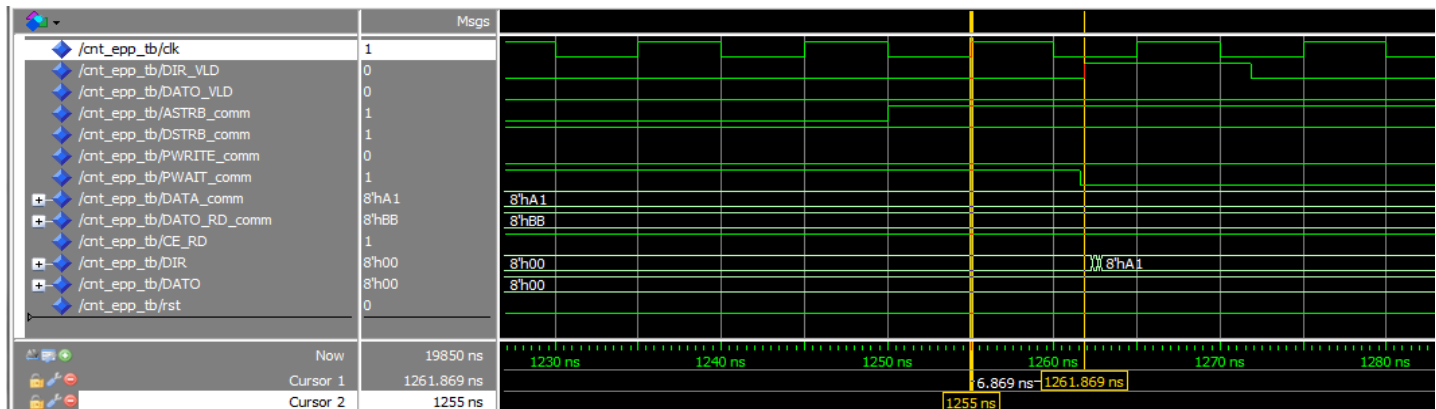
Las tres barras divisoras de color azul de la figura muestran los momentos de escritura. Se puede ver como el primer caso escribe en la dirección A1h (*dir_dpram1*) el dato 34h, la segunda división escribe en la posición A2h (*dir_dpram2*) el dato 44h y el tercer divisor azul escribe en la posición F0h (*dir_frec*) el dato 54. Por último el divisor de color rojo, indica un ciclo de lectura, en este caso lee de la posición FFh.

1.2.1.- Tabla donde se incluya los recursos utilizados para la implementación del controlador del puerto EPP. Los datos anteriores serán extraídos del informe de síntesis.

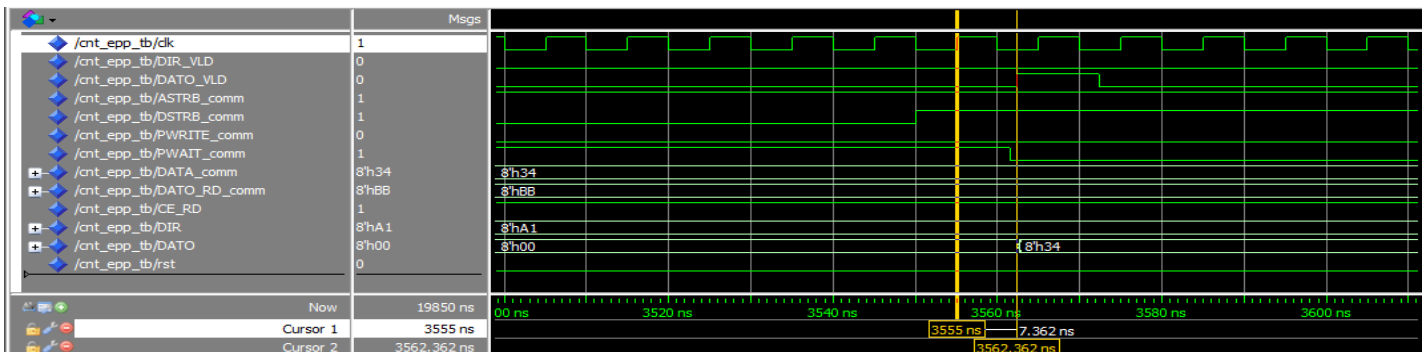
Device Utilization Summary				
Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	21	54,576	1%	
Number used as Flip Flops	21			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	20	27,288	1%	
Number used as logic	20	27,288	1%	
Number using O6 output only	19			
Number using O5 output only	0			
Number using O5 and O6	1			
Number used as ROM	0			
Number used as Memory	0	6,408	0%	

Number of occupied Slices	8	6,822	1%	
Number of MUXCYs used	0	13,644	0%	
Number of LUT Flip Flop pairs used	20			
Number with an unused Flip Flop	1	20	5%	
Number with an unused LUT	0	20	0%	
Number of fully used LUT-FF pairs	19	20	95%	
Number of unique control sets	1			
Number of slice register sites lost to control set restrictions	3	54,576	1%	
Number of bonded IOBs	41	218	18%	
IOB Flip Flops	1			
Number of RAMB16BWERs	0	116	0%	
Number of RAMB8BWERs	0	232	0%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%	
Number of BUFG/BUFGMUXs	1	16	6%	
Number used as BUFGs	1			
Number used as BUFGMUX	0			
Number of DCM/DCM_CLKGENs	0	8	0%	
Number of ILOGIC2/ISERDES2s	0	376	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	376	0%	
Number of OLOGIC2/OSERDES2s	1	376	1%	
Number used as OLOGIC2s	1			
Number used as OSERDES2s	0			
Number of BSCANs	0	4	0%	
Number of BUFHs	0	256	0%	
Number of BUFPLLs	0	8	0%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	58	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	0	4	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	3.36			

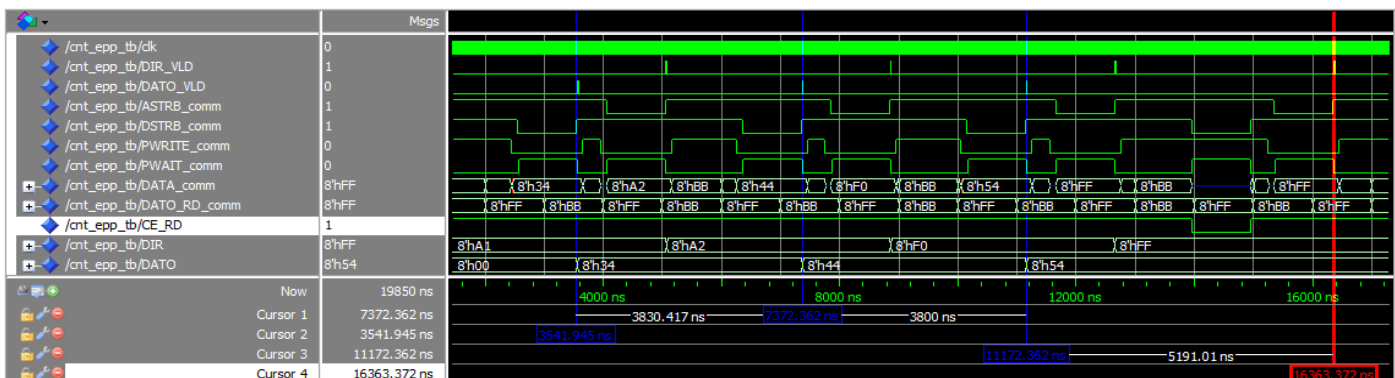
1.2.2.- Medida del retardo entre el flanco activo de la señal de reloj y la activación de los puertos DIR_VLD y DATO_VLD. Se debe mostrar el resultado a través de pantallazos de simulación utilizando, para ello, dos cursores.



La figura anterior muestra el retardo entre el flanco activo de la señal de reloj y la activación del puerto DIR_VLD, como se aprecia hay un cierto retardo de 6.869 ns. En el siguiente caso se muestra el retardo entre el flanco activo del reloj y DATO_VLD el cual es de 7.362 ns, un poco mayor que el caso anterior.



1.2.3.- Pantallazo de simulación donde se refleje el que la entidad cnt_epp función correctamente.



En la simulación temporal se puede apreciar la misma conducta que ocurriría en la simulación funcional, luego se deduce que el comportamiento sigue siendo correcto, sólo que en esta ocasión se incluyen retardos provocados por el hardware.

2. TOP SYSTEM1

2.1.1.- Especificación en VHDL de los modelos *top_system1* y el correspondiente banco de pruebas *top_sytem1_tb*.

A) Código VHDL de la entidad *top_system1*:

```
entity top_system1 is
port (
  CLK          : in  std_logic;
  RST          : in  std_logic;
  ASTRB        : in  std_logic;
  DSTRB        : in  std_logic;
  DATA        : inout std_logic_vector(7 downto 0);
  PWRITE       : in  std_logic;
  PWAIT        : out  std_logic;
  SWITCHES_I   : in  std_logic_vector(7 downto 0);
  PSH_BUTTON   : in  std_logic;
  LEDS_O       : out  std_logic_vector(7 downto 0);
);

end top_system1;

architecture Behavioral of top_system1 is

  component cnt_epp
  port (
    CLK   : in  std_logic;
    RST   : in  std_logic;
    ASTRB : in  std_logic;
    DSTRB : in  std_logic;
    DATA : inout std_logic_vector(7 downto 0);
    PWRITE : in  std_logic;
    PWAIT : out  std_logic;
    DATO_RD : in  std_logic_vector(7 downto 0);
    CE_RD   : out  std_logic;
    DIR     : out  std_logic_vector (7 downto 0);
    DIR_VLD : out  std_logic;
    DATO    : out  std_logic_vector (7 downto 0);
    DATO_VLD : out  std_logic;
  end component ;

  signal DATO_RD_comm      : std_logic_vector(7 downto 0);
  signal CE_RD_comm       : std_logic;
  signal DIR_comm         : std_logic_vector (7 downto 0);
  signal DIR_VLD_comm     : std_logic;
  signal DATO_comm        : std_logic_vector (7 downto 0);
  signal DATO_VLD_comm    : std_logic;
  signal DIR_REG          : std_logic_vector (7 downto 0);
  signal DATO_REG         : std_logic_vector (7 downto 0);
```

```
begin
  cntEpp:cnt_epp
    port map(
      CLK=>CLK ,
      RST=>RST ,
      ASTRB=>ASTRB ,
      DSTRB=>DSTRB ,
      DATA=>DATA ,
      PWRITE=>PWRITE ,
      PWAIT=>PWAIT,
      DATO_RD=>DATO_RD_comm,
      CE_RD=>CE_RD_comm,
      DIR=>DIR_comm,
      DIR_VLD=>DIR_VLD_comm,
      DATO=>DATO_comm,
      DATO_VLD=>DATO_VLD_comm
    );

  AddrBiestableD:process (CLK,RST,DIR_VLD_comm)
  begin
    if (RST='1') then
      DIR_REG <= (others=>'0');
    elsif (CLK'event and CLK='1')then
      if (DIR_VLD_comm='1') then
        DIR_REG <= DIR_comm;
      end if;
    end if;
  end process;

  DataBiestableD:process (CLK,RST,DATO_VLD_comm)
  begin
    if (RST='1') then
      DATO_REG <= (others=>'0');
    elsif (CLK'event and CLK='1')then
      if (DATO_VLD_comm='1') then
        DATO_REG <= DATO_comm;
      end if;
    end if;
  end process;

  LEDS_O <= DATO_REG when (PSH_BUTTON='0') else DIR_REG;

  CE_RDbiestableD:process (CLK,RST)
  begin
    if (RST='1') then
      DATO_RD_comm <= (others=>'0');
    elsif (CLK'event and CLK='1')then
      if (CE_RD_comm='1' and DIR_comm=x"32") then
        DATO_RD_comm <= SWITCHES_I;
      end if;
    end if;
  end process;
end Behavioral;
```

B) Código VHDL del banco de pruebas la entidad **top_system1_tb**:

```
ENTITY top_system1_tb IS
END top_system1_tb;

ARCHITECTURE behavior OF top_system1_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT top_system1
    PORT(
        CLK : IN std_logic;
        RST : IN std_logic;
        ASTRB : IN std_logic;
        DSTRB : IN std_logic;
        DATA : INOUT std_logic_vector(7 downto 0);
        PWRITE : IN std_logic;
        PWAIT : OUT std_logic;
        SWITCHES_I : IN std_logic_vector(7 downto 0);
        PSH_BUTTON : IN std_logic;
        LEDS_O : OUT std_logic_vector(7 downto 0)
    );
    END COMPONENT;

    COMPONENT epp_device
    port (
        DATA : inout std_logic_vector(7 downto 0);
        PWRITE : out std_logic;
        DSTRB : out std_logic;
        ASTRB : out std_logic;
        PWAIT : in std_logic;
    )
    END COMPONENT;

    signal ASTRB_comm:std_logic;
    signal DSTRB_comm:std_logic;
    signal PWRITE_comm:std_logic;
    signal PWAIT_comm:std_logic;
    signal DATA_comm:std_logic_vector(7 downto 0);

    signal SWITCHES_I_comm : std_logic_vector(7 downto 0) := x"CA";
    signal PSH_BUTTON_comm : std_logic := '0';
    signal LEDS_O_comm : std_logic_vector(7 downto 0);

    -- Clock period definitions
    constant CLK_period : time := 10 ns;
    signal CLK : std_logic;
    signal RST : std_logic:='1';

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: top_system1 PORT MAP (
        CLK => CLK,
        RST => RST,
        ASTRB => ASTRB_comm,
        DSTRB => DSTRB_comm,
        DATA => DATA_comm,
        PWRITE => PWRITE_comm,
        PWAIT => PWAIT_comm,
        SWITCHES_I => SWITCHES_I_comm,
        PSH_BUTTON => PSH_BUTTON_comm,
        LEDS_O => LEDS_O_comm
    );
```

```
eppDevice:epp_device
    port map(
        DATA=>DATA_comm,
        PWRITE=>PWRITE_comm,
        DSTRB=>DSTRB_comm,
        ASTRB=>ASTRB_comm,
        PWAIT=>PWAIT_comm
    );

    -- Clock process definitions
    CLK_process :process
    begin
        CLK <= '0';
        wait for CLK_period/2;
        CLK <= '1';
        wait for CLK_period/2;
    end process;

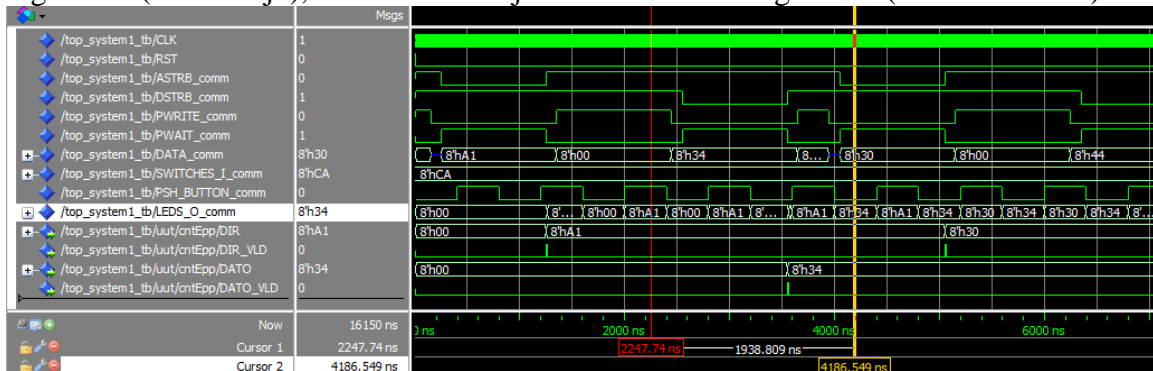
    RST_process :process
    begin
        wait for 5 ns;
        RST <= '0';
    end process;

    PSH_BUTTON_process :process
    begin
        PSH_BUTTON_comm <= '0';
        wait for 400ns;
        PSH_BUTTON_comm <= '1';
        wait for 400ns;
    end process;

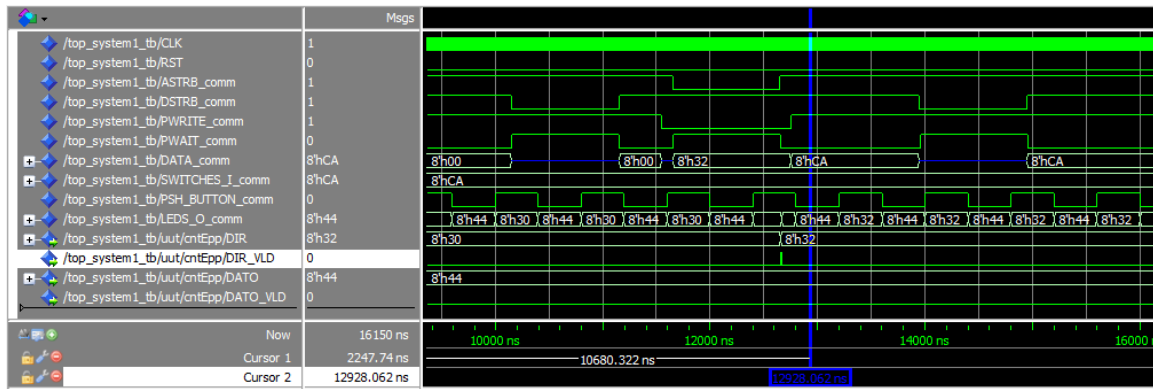
END;
```

2.1.2.- Pantallazos de la simulación funcional de los modelos anteriores donde se demuestre que funcionan correctamente.

En el esquema cuando PUSH_BUTTON está a nivel alto los leds muestran la dirección registrada (cursor rojo), con un nivel bajo muestra el dato registrado (cursor amarillo).

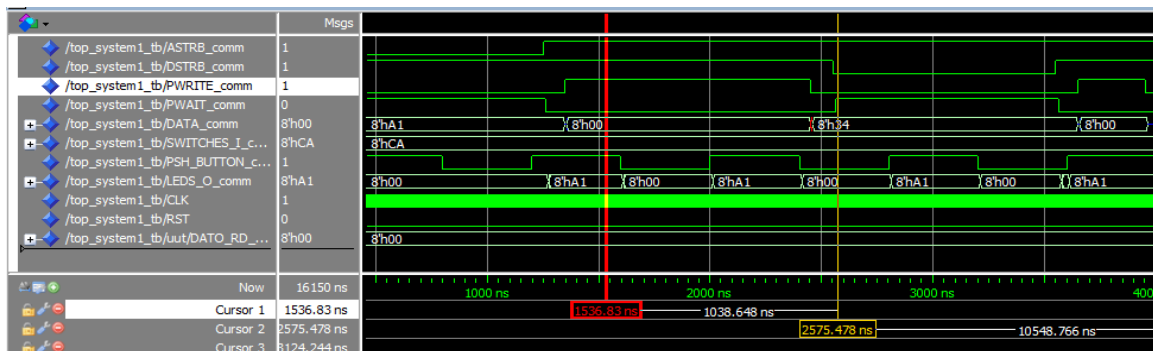


La entrada de los 8 switches programados en el testbench con un valor de CAh pasara a DATO_RD cuando se realice una lectura de la posición 32h (cursor azul)

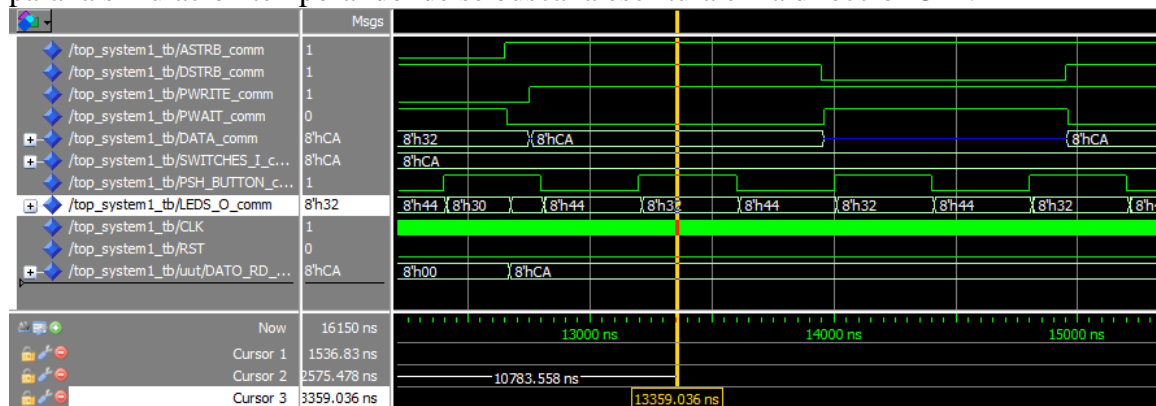


2.1.3.- Pantallazos de la simulación temporal de los modelos anteriores donde se demuestre que funcionan correctamente. En este caso se deberá añadir una tabla donde se reflejen los recursos utilizados en su implementación.

A) Pantallazos simulación temporal



Esta tabla muestra la simulación temporal y como se siguen cumpliendo los eventos que se cumplían en la simulación funcional anteriormente explicada. De igual manera sucede para la simulación temporal donde se busca la escritura en la dirección 32h.



B) Tabla donde se incluya los recursos utilizados para la implementación del controlador del puerto EPP. Los datos anteriores serán extraídos del informe de síntesis.

Device Utilization Summary					
Slice Logic Utilization	Used	Available	Utilization	Note(s)	
Number of Slice Registers	45	54,576	1%		
Number used as Flip Flops	45				
Number used as Latches	0				
Number used as Latch-thrus	0				
Number used as AND/OR logics	0				
Number of Slice LUTs	13	27,288	1%		
Number used as logic	9	27,288	1%		
Number using O6 output only	3				
Number using O5 output only	0				
Number using O5 and O6	6				
Number used as ROM	0				
Number used as Memory	0	6,408	0%		
Number used exclusively as route-thrus	4				
Number with same-slice register load	4				
Number with same-slice carry load	0				
Number with other load	0				
Number of occupied Slices	14	6,822	1%		
Number of MUXCYs used	0	13,644	0%		
Number of LUT Flip Flop pairs used	43				
Number with an unused Flip Flop	2	43	4%		
Number with an unused LUT	30	43	69%		
Number of fully used LUT-FF pairs	11	43	25%		
Number of unique control sets	6				
Number of slice register sites lost to control set restrictions	3	54,576	1%		
Number of bonded IOBs	31	218	14%		

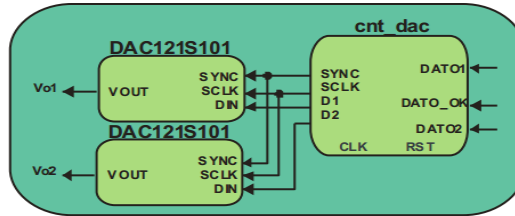
Number of LOCed IOBs	31	31	100%	
IOB Flip Flops	1			
Number of RAMB16BWERs	0	116	0%	
Number of RAMB8BWERs	0	232	0%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%	
Number of BUFG/BUFGMUXs	1	16	6%	
Number used as BUFGs	1			
Number used as BUFGMUX	0			
Number of DCM/DCM_CLKGENs	0	8	0%	
Number of ILOGIC2/ISERDES2s	0	376	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	376	0%	
Number of OLOGIC2/OSERDES2s	1	376	1%	
Number used as OLOGIC2s	1			
Number used as OSERDES2s	0			
Number of BSCANs	0	4	0%	
Number of BUFHs	0	256	0%	
Number of BUFPLLs	0	8	0%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	58	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	0	4	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	2.39			

2.2.1.- Pantallazos como el de la Figura 28 donde se refleje que la entidad top_system1 correctamente para las distintas operaciones.

CAPTURA DIGILENT 1	CAPTURA DIGILENT 2
--------------------	--------------------

3. CNT_DAC

3.1.1.- Código VHDL del módulo de control de los DACs y del banco de pruebas utilizado en su simulación.



A) Código VHDL de la entidad **cnt_dac**:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity cnt_dac is
port (
    CLK : in std_logic;
    RST : in std_logic;
    DATO1 : in std_logic_vector(7 downto 0);
    DATO2 : in std_logic_vector(7 downto 0);
    DATO_OK : in std_logic;
    SYNC : out std_logic;
    SCLK : out std_logic;
    D1 : out std_logic;
    D2 : out std_logic);
end cnt_dac;

architecture RTL of cnt_dac is

type state_type is (
    s0, --HOLDING STATE
    s1, --TRANSMISSION STATE
    s2 --WAITING STATE
);
signal state: state_type; --current and next state declaration.
type mux_fsm is (
    mx0, --HOLDING STATE
    mx1,mx2
);
signal mxstate: mux_fsm; --current and next state declaration.
signal SCLKaux: std_logic;
signal muxSelect: integer range 0 to 3;
signal muxSelectTX: unsigned (3 downto 0);
signal CEcounter: std_logic;
signal DATO_1_16bits: std_logic_vector(15 downto 0);
signal DATO_2_16bits: std_logic_vector(15 downto 0);
signal endTx: std_logic;
constant countLimit : integer:=64;
constant sclkPulseValue : unsigned:="1";
signal sclkPulseCounter: integer range 0 to 1;
signal prescaler : unsigned(0 downto 0);
BEGIN
    DATO_1_16bits<="0000"&DATO1&"0000";
    DATO_2_16bits<="0000"&DATO2&"0000";
```

```
gen_clk : process (clk, rst)
begin -- process gen_clk
    if rst = '1' then
        SCLKaux <= '0';
        prescaler <= (others => '0');
    elsif (clk'event and clk='1') then -- rising clock edge
        if prescaler = X"1" then
            prescaler <= (others => '0');
            SCLKaux <= not SCLKaux;
        else
            prescaler <= prescaler + "1";
        end if;
    end if;
end process gen_clk;

SCLK <= SCLKaux;

counter:process(rst,CEcounter,clk)
begin
    if(rst = '1') then
        muxSelect <= 0;
        muxSelectTX<=(others=>'0');
    elsif (clk'event and clk='1') then
        if (CEcounter='1') then
            if (muxSelect = 3 ) then
                muxSelect <= 0;
                muxSelectTX<=muxSelectTX+1;
            else
                muxSelect <= muxSelect+1;
            end if;
        else
            muxSelectTX<=(others=>'0');
            muxSelect <= 0;
        end if;
    end if;
end process;
```

```

mux:process (muxSelectTX,DATO_1_16bits,DATO_2_16bits)
begin
    endTx <= '0';
case muxSelectTX is
when x"0"=> D1 <= DATO_1_16bits(15);
              D2 <= DATO_2_16bits(15);
when x"1"=> D1 <= DATO_1_16bits(14);
              D2 <= DATO_2_16bits(14);
when x"2"=> D1 <= DATO_1_16bits(13);
              D2 <= DATO_2_16bits(13);
when x"3"=> D1 <= DATO_1_16bits(12);
              D2 <= DATO_2_16bits(12);
when x"4"=> D1 <= DATO_1_16bits(11);
              D2 <= DATO_2_16bits(11);
when x"5"=> D1 <= DATO_1_16bits(10);
              D2 <= DATO_2_16bits(10);
when x"6"=> D1 <= DATO_1_16bits(9);
              D2 <= DATO_2_16bits(9);
when x"7"=> D1 <= DATO_1_16bits(8);
              D2 <= DATO_2_16bits(8);
when x"8"=> D1 <= DATO_1_16bits(7);
              D2 <= DATO_2_16bits(7);
when x"9"=> D1 <= DATO_1_16bits(6);
              D2 <= DATO_2_16bits(6);
when x"A"=> D1 <= DATO_1_16bits(5);
              D2 <= DATO_2_16bits(5);
when x"B"=> D1 <= DATO_1_16bits(4);
              D2 <= DATO_2_16bits(4);
when x"C"=> D1 <= DATO_1_16bits(3);
              D2 <= DATO_2_16bits(3);
when x"D"=> D1 <= DATO_1_16bits(2);
              D2 <= DATO_2_16bits(2);
when x"E"=> D1 <= DATO_1_16bits(1);
              D2 <= DATO_2_16bits(1);
when x"F"=> D1 <= DATO_1_16bits(0);
              D2 <= DATO_2_16bits(0);
              endTx <= '1';
when others => D1 <= '0';
               D2 <= '0';
end case;
end process;

```

```

fsm_proc:process(clk, rst,endTx,DATO_OK)
begin
if(rst = '1') then
state <= S0;
SYNC<='1';
CEcounter<='0';
elsif (clk'event and clk = '1') then
case state is
when s0 =>
if (DATO_OK='0') then
SYNC<='1';
CEcounter<='0';
else --if DATO_OK=1
state <= s1;
end if;
when s1 =>
CEcounter<='1';
SYNC<='0';
if(endTx = '1') then
CEcounter<='0';
SYNC<='1';
state <= s2;
end if;
when s2 =>
SYNC<='1';
state <= s0;
end case;
end if;
end process;
smMux_proc:process(clk, rst,muxSelectTX)
begin
if(rst = '1') then
endTx <= '0';
mxstate <= mx0;
elsif (clk'event and clk = '1') then
case mxstate is
when mx0 =>
endTx <= '0';
if (muxSelectTX=x"F") then
mxstate <= mx1;
end if;
when mx1 =>
if (muxSelectTX=x"F") then
mxstate <= mx2;
else
mxstate <= mx0;
end if;
when mx2 =>
mxstate <= mx0;
endTx <= '1';
end case;
end if;
end process;
end RTL;

```


B) Código VHDL del testbench de la entidad *cnt_dac_tb*:

```
ENTITY cnt_dac_tb IS
END cnt_dac_tb;

ARCHITECTURE behavior OF cnt_dac_tb IS
    component cnt_dac
    port (
        CLK    : in  std_logic;
        RST    : in  std_logic;
        DATO1   : in  std_logic_vector(7 downto 0);
        DATO2   : in  std_logic_vector(7 downto 0);
        DATO_OK : in  std_logic;
        SYNC    : out std_logic;
        SCLK    : out std_logic;
        D1      : out std_logic;
        D2      : out std_logic);
    end component ;

    component DAC121S101
    port (
        VOUT : out real range 0.0 to 3.5;
        SYNC : in  std_logic;
        SCLK : in  std_logic;
        DIN  : in  std_logic);
    end component ;

    --Inputs
    signal CLK : std_logic := '0';
    signal RST : std_logic := '1';
    signal DATO1 : std_logic_vector(7 downto 0):=x"00";
    signal DATO2 : std_logic_vector(7 downto 0):=x"00";
    signal DATO_OK : std_logic := '0';

    --Outputs
    signal VOUT1 : real;
    signal VOUT2 : real;
    signal SYNC_comm      : std_logic;
    signal SCLK_comm      : std_logic;
    signal D1_comm        : std_logic;
    signal D2_comm        : std_logic;

    -- Clock period definitions
    constant CLK_period : time := 10 ns;
BEGIN
    cntdac:cnt_dac
        port map(
            CLK=>CLK ,
            RST=>RST ,
            DATO1=>DATO1 ,
            DATO2=>DATO2 ,
            DATO_OK=>DATO_OK,
            SYNC=>SYNC_comm ,
            SCLK=>SCLK_comm,
            D1=>D1_comm,
            D2=>D2_comm
        );

    DAC1: DAC121S101
        port map (
            VOUT =>VOUT1,
            SYNC =>SYNC_comm,
            SCLK =>SCLK_comm,
            DIN  => D1_comm);

    DAC2: DAC121S101
        port map (
            VOUT =>VOUT2,
            SYNC =>SYNC_comm,
            SCLK =>SCLK_comm,
            DIN => D2_comm
        );
end;
```

```
clk_process :process
begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
end process;

rst_process :process
begin
    wait for 100 ns;
    rst <= '0';
end process;

-- Stimulus process
stim_proc: process
begin
    -- hold reset state for 100 ns.
    wait for 800 ns;
    DATO1 <=std_logic_vector(to_unsigned(5, 8));
    DATO2 <=std_logic_vector(to_unsigned(10, 8));
    DATO_OK<='1';
        wait for 20 ns;
    DATO_OK<='0';
        wait for 800 ns;
    DATO1 <=std_logic_vector(to_unsigned(30, 8));
    DATO2 <=std_logic_vector(to_unsigned(40, 8));
    DATO_OK<='1';
        wait for 20 ns;
    DATO_OK<='0';
        wait for 800 ns;
    DATO1 <=std_logic_vector(to_unsigned(85, 8));
    DATO2 <=std_logic_vector(to_unsigned(95, 8));
    DATO_OK<='1';
        wait for 20 ns;
    DATO_OK<='0';
        wait;

end process;

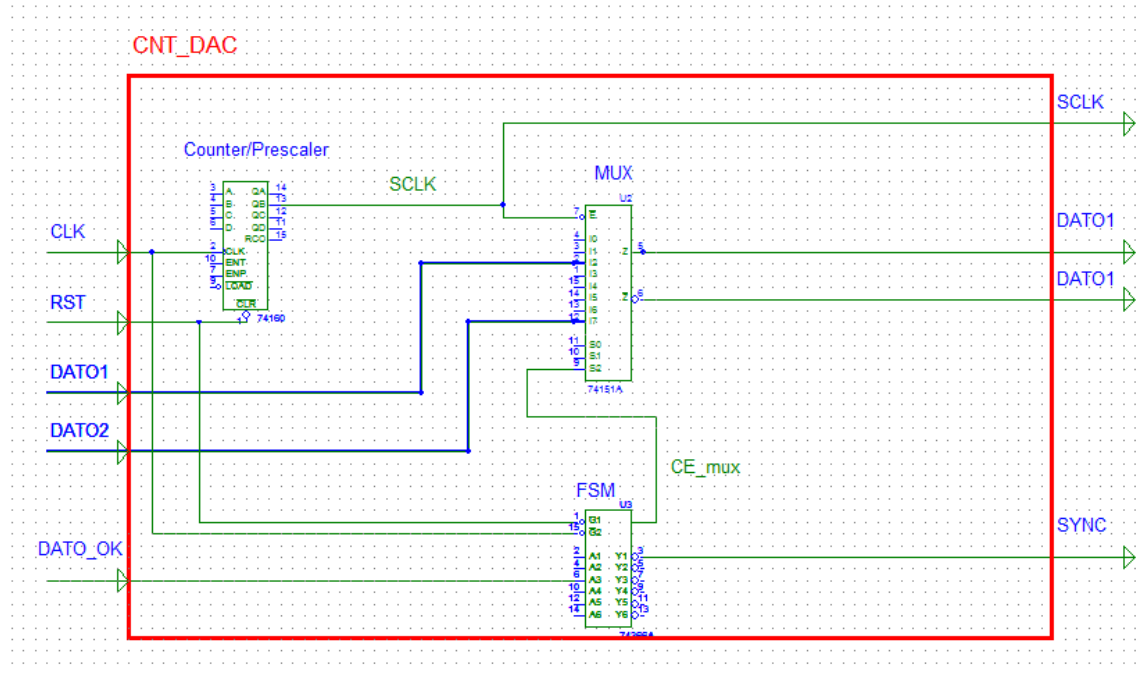
END;
```

3.1.2- Razonamiento de por qué se ha adoptado la solución presentada.

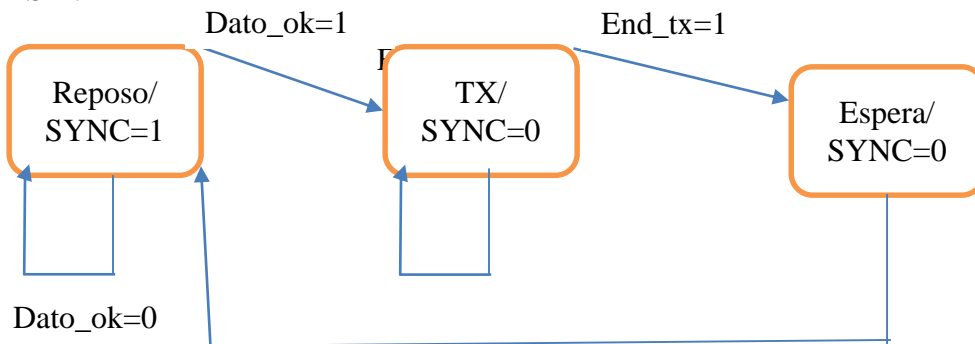
En el diseño de este módulo se han creado dos contadores, un modificador de frecuencia de reloj un multiplexor y dos máquinas de estados.

El primer contador se usa para contar flancos de subida de reloj. La salida del contador es usada por el modificador de frecuencia para crear la señal SCLK.

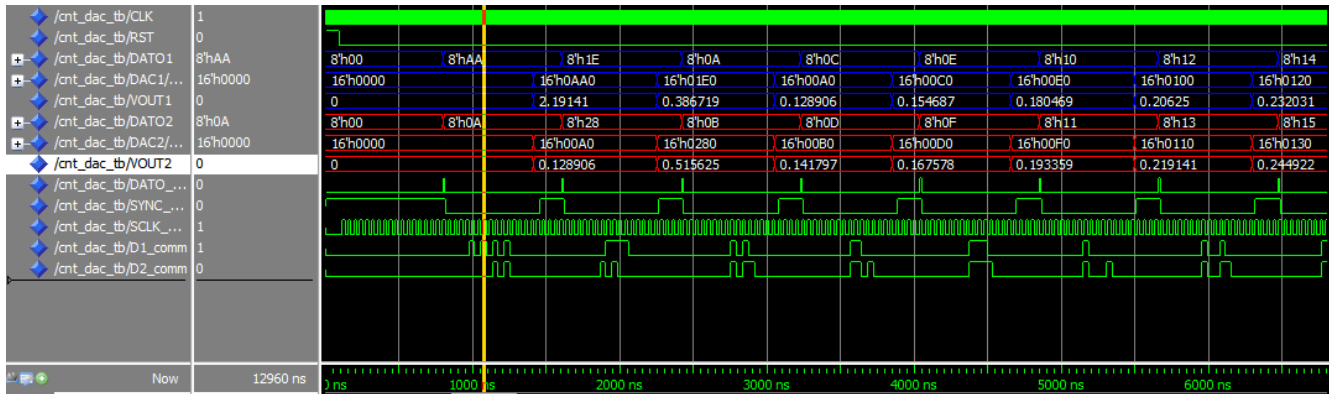
El segundo contador es usado para incrementar la cuenta de selección del multiplexor. El multiplexor es el encargado de serializar en la salida el valor del dato de entrada.



FSM:



3.1.3.- Pantallazo de simulación funcional donde se refleje que le mencionado circuito funciona correctamente. Para ello en el banco de pruebas deberá incluir los estímulos necesarios para comprobar que para todas las operaciones el módulo funciona correctamente.



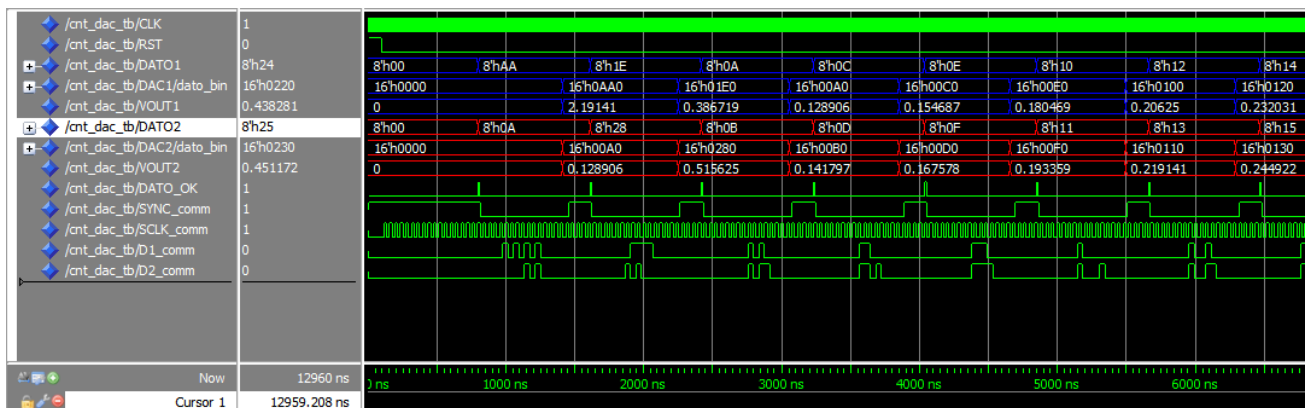
En el esquema anterior se ve como los datos de entrada DATO1 y DATO2 son serializados en D1_comm y D2_comm y tras pasar por los DAC121S101 la salida en se produce por VOUT con un valor decimal analógico. El paso de datos del cnt_dac a los DAC se pueden ver con los valores de dato_bin que coincide con la entrada de datos de cnt_dac.

3.2.1.- Tabla donde se incluyan los recursos utilizados para la implementación.

Device Utilization Summary				
Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	9	54,576	1%	
Number used as Flip Flops	9			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	15	27,288	1%	
Number used as logic	15	27,288	1%	
Number using O6 output only	13			
Number using O5 output only	0			
Number using O5 and O6	2			
Number used as ROM	0			
Number used as Memory	0	6,408	0%	
Number of occupied Slices	7	6,822	1%	
Number of MUXCYs used	0	13,644	0%	
Number of LUT Flip Flop pairs used	15			
Number with an unused Flip Flop	6	15	40%	
Number with an unused LUT	0	15	0%	
Number of fully used LUT-FF pairs	9	15	60%	
Number of unique control sets	2			

Number of slice register sites lost to control set restrictions	7	54,576	1%	
Number of bonded IOBs	23	218	10%	
Number of RAMB16BWERs	0	116	0%	
Number of RAMB8BWERs	0	232	0%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%	
Number of BUFG/BUFGMUXs	1	16	6%	
Number used as BUFGs	1			
Number used as BUFGMUX	0			
Number of DCM/DCM_CLKGENs	0	8	0%	
Number of ILOGIC2/ISERDES2s	0	376	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	376	0%	
Number of OLOGIC2/OSERDES2s	0	376	0%	
Number of BSCANs	0	4	0%	
Number of BUFHs	0	256	0%	
Number of BUFPLLs	0	8	0%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	58	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	0	4	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	2.29			

3.2.2.- Pantallazo de simulación donde se refleje que el controlador de los DACs funciona correctamente.



La anterior grafica muestra un comportamiento similar a la simulación funcional, pero se ha tenido que modificar el tiempo del pulso de DATO_OK ya que en esta simulación con 10 ns no tenía tiempo para modificar valores.

4. DPRAM_MEM

4.1.1.- Código VHDL del módulo y del banco de pruebas utilizado en la simulación.

A) Código VHDL de la entidad ***dpram_mem***

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dpram_mem is
  port (
    DIN      : in  std_logic_vector(7 downto 0);
    ADDR_IN  : in  std_logic_vector(7 downto 0);
    ADDR_OUT : in  std_logic_vector(7 downto 0);
    DOUT     : out std_logic_vector(7 downto 0);
    WE       : in  std_logic;
    CLK      : in  std_logic;
    RST      : in  std_logic;
  );
end entity;

architecture rtl of dpram_mem is

  type memory is array (256 downto 0)
  of std_logic_vector(7 downto 0);
  signal mem_pos: memory;
begin
  process(clk)
  begin
    if (clk'event and clk = '1') then
      if (WE = '1') then
        mem_pos(to_integer(unsigned(ADDR_IN))) <= DIN;
      end if;
    end if;
  end process;

  process(clk)
  begin
    if (clk'event and clk = '1') then
      DOUT <= mem_pos(to_integer(unsigned(ADDR_OUT)));
    end if;
  end process;

end rtl;
```

B) Código VHDL testbench ***dpram_mem_tb***

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

ENTITY dpram_mem_tb IS
END dpram_mem_tb;

ARCHITECTURE behavior OF dpram_mem_tb IS
  COMPONENT dpram_mem
  PORT(
    DIN : IN  std_logic_vector(7 downto 0);
    ADDR_IN : IN std_logic_vector(7 downto 0);
    ADDR_OUT : IN std_logic_vector(7 downto 0);
    DOUT : OUT std_logic_vector(7 downto 0);
    WE : IN  std_logic;
    CLK : IN  std_logic;
    RST : IN  std_logic
  );
  END COMPONENT;

  signal DIN : std_logic_vector(7 downto 0) := (others => '0');
  signal ADDR_IN : std_logic_vector(7 downto 0) := (others => '0');
  signal ADDR_OUT : std_logic_vector(7 downto 0) := (others => '0');
  signal WE : std_logic := '0';
  signal CLK : std_logic := '0';
  signal RST : std_logic := '1';
  signal pulseCounter: integer range 1 to 5;
  signal DOUT : std_logic_vector(7 downto 0);
  constant CLK_period : time := 10 ns;
BEGIN
  uut: dpram_mem PORT MAP (
    DIN => DIN,
    ADDR_IN => ADDR_IN,
    ADDR_OUT => ADDR_OUT,
    DOUT => DOUT,
    WE => WE,
    CLK => CLK,
    RST => RST
  );

  clk_process : process
  begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
  end process;

  rst_process : process
  begin
    wait for 100 ns;
    rst <= '0';
  end process;

  we_process : process
  begin
    we <= '1';
    wait for 10 ns;
    we <= '0';
    wait for 10 ns;
  end process;
```

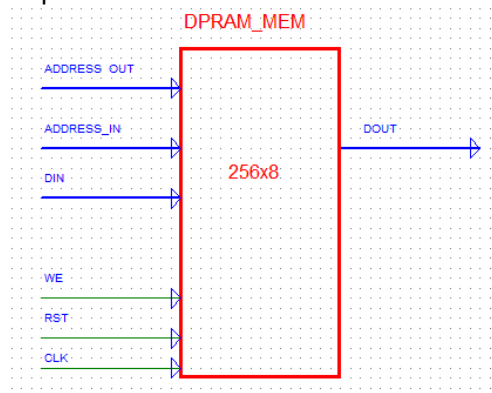
B) Continuación testbench *dpram_mem_tb*

```
din_process:process (we,rst)
begin
    if (rst='1') then
        DIN <= x"09";
    elsif (we'event and we='1')then
        DIN<=std_logic_vector(unsigned(DIN) + x"01");
    end if;
end process;
addressIn_process:process (we,rst)
begin
    if (rst='1') then
        ADDR_IN <= x"00";
    elsif (we'event and we='1')then
        ADDR_IN<=std_logic_vector(unsigned(ADDR_IN) + x"01");
    end if;
end process;
addressOut_process:process (clk,rst)
begin
    if (rst='1') then
        ADDR_OUT <= x"00";
    elsif (clk'event and clk='0')then
        if(pulseCounter = 5)then
            pulseCounter <= 1;
            ADDR_OUT<=std_logic_vector(unsigned(ADDR_OUT) + x"01");
        else
            pulseCounter<=pulseCounter+1;
        end if;
    end if;
end process;
END;
```

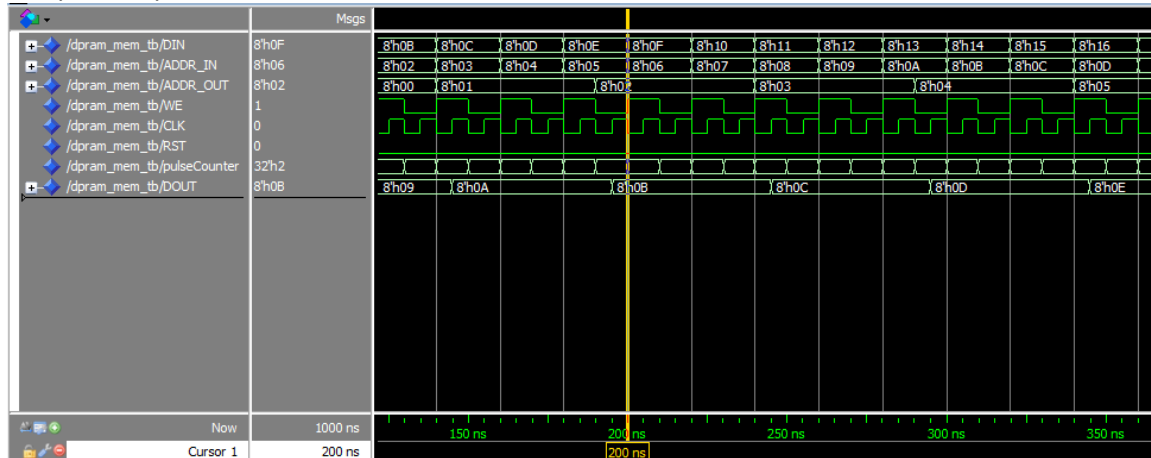
4.1.2- Estudio detallado del funcionamiento de la memoria dual port.

La memoria dual port para el ciclo de escritura recibe datos por el puerto DIN y direcciones por el puerto ADDR_IN. Cuando se activa la señal WE el dato recibido se almacena en un array bidimensional de 256 direcciones de 8 bits cada dirección, en la dirección específica por ADDR_IN.

Para el ciclo de lectura recibe la dirección a través del puerto ADDR_OUT y DOUT envía el dato almacenado en el array correspondiente a esa dirección.



4.1.3.- Pantallazo de simulación donde se refleje que el mencionado circuito funciona correctamente. Para ello en el banco de pruebas deberá incluir los estímulos necesarios para comprobar que el módulo funciona correctamente.



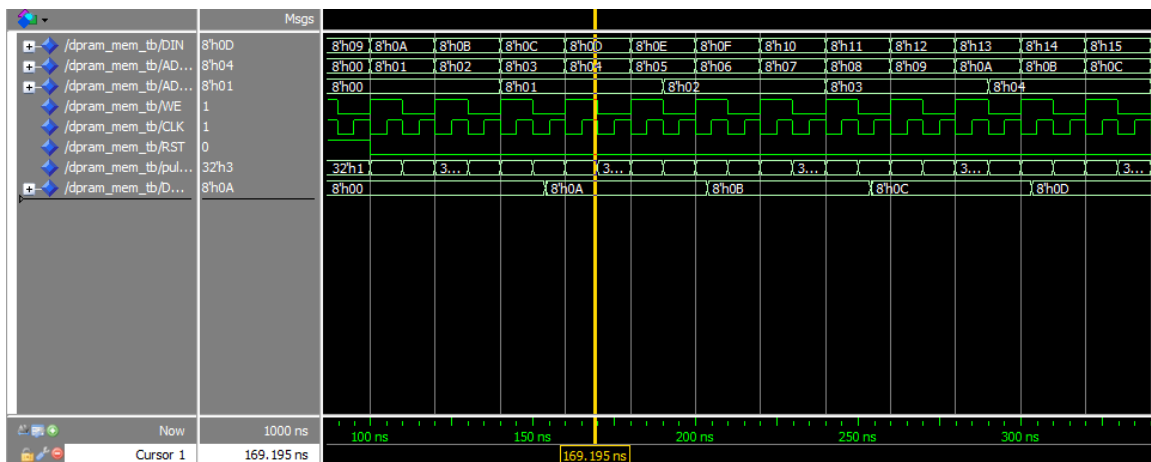
La gráfica anterior muestra la simulación funcional de la memoria RAM *dpram_mem*. En él se puede ver como a la salida DOUT tiene el dato que se ha introducido en ADDR_IN pero la salida se produce con un desplazamiento en el tiempo respecto a la entrada.

4.2.1.- Tabla donde se incluyan los recursos utilizados para la implementación.

Device Utilization Summary					
Slice Logic Utilization	Used	Available	Utilization	Note(s)	
Number of Slice Registers	0	54,576	0%		
Number of Slice LUTs	0	27,288	0%		
Number of occupied Slices	0	6,822	0%		
Number of MUXCYs used	0	13,644	0%		
Number of LUT Flip Flop pairs used	0				
Number of bonded IOBs	34	218	15%		
Number of RAMB16BWERs	0	116	0%		
Number of RAMB8BWERs	1	232	1%		
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%		
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%		
Number of BUFG/BUFGMUXs	1	16	6%		
Number used as BUFGs	1				
Number used as BUFGMUX	0				
Number of DCM/DCM_CLKGENs	0	8	0%		
Number of ILOGIC2/ISERDES2s	0	376	0%		
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	376	0%		
Number of OLOGIC2/OSERDES2s	0	376	0%		
Number of BSCANs	0	4	0%		
Number of BUFHs	0	256	0%		

Number of BUFPLLs	0	8	0%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	58	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCIOLOGICSEs	0	2	0%	
Number of PLL_ADVs	0	4	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	1.03			

4.2.2.- Pantallazo de simulación donde se refleje que el módulo funciona correctamente.



Como se puede ver en la simulación temporal, el comportamiento de la memoria es igual a la simulación funcional pero con algunos retardos en la inicialización y en la transición de las señales provocadas por el hardware.

5. CNT DPRAM

5.1.1.- Código VHDL del módulo y del banco de pruebas utilizado en la simulación.

A) Código VHDL de la entidad *cnt_dpram*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity cnt_dpram is
port (
    CLK    : in std_logic;
    RST    : in std_logic;
    DIR    : in std_logic_vector (7 downto 0);
    DIR_VLD : in std_logic;
    DATO   : in std_logic_vector (7 downto 0);
    DATO_VLD : in std_logic;
    ADDRESS : out std_logic_vector(7 downto 0);
    DATA  : out std_logic_vector(7 downto 0);
    WE_DP1 : out std_logic;
    WE_DP2 : out std_logic);
end cnt_dpram;

architecture RTL of cnt_dpram is
    constant dir_dpram1 : std_logic_vector( 7 downto 0) := x"A1";
    constant dir_dpram2 : std_logic_vector( 7 downto 0) := x"A2";
    type state_type is (
        s0, --HOLDING STATE
        s1, --WAIT DATA STATE
        s2, --CLOCK ENABLE STATE
        s3, --CLEAR STATE
        s4, --WE_DP ENABLE STATE
    );

    signal state:          state_type
    signal dirActual: std_logic_vector(7 downto 0);
    signal dirPrev:      std_logic_vector(7 downto 0);
    signal CCounter: std_logic;
    signal clearCounter: std_logic;
    signal addressTX: std_logic_vector(7 downto 0);
    signal counter: std_logic_vector(7 downto 0);

    begin -- RTL
        DATA<=DATO;
        counter_proc:process(clk,rst,CCounter,clearCounter)
        begin
            if (rst='1')then
                counter<=(others=>'0');
            elsif (clk'event and clk = '1') then
                if(CCounter='1') then
                    if(counter=x"FF")then
                        counter<=x"00";
                    else
                        counter<=std_logic_vector(unsigned(counter) + 1);
                    end if;
                end if;
            elsif (clearCounter='1') then
                counter<=(others=>'0');
            end if;
        end if;
    end process;
end cnt_dpram;
```

```
ADDRESS<=counter;
fsm_proc:process(clk, rst,DIR_VLD,DIR,DATO_VLD,DATO)
begin
    if(rst = '1') then
        state <= s0;
        CCounter<='0';
        we_dp1<='0';
        we_dp2<='0';
        dirPrev<=(others=>'0');
        dirActual<=(others=>'0');
        clearCounter<='0';
    elsif (clk'event and clk = '1') then
        case state is
            WHEN s0 =>

                we_dp1<='0';
                we_dp2<='0';
                clearCounter<='0';
                dirActual<=DIR;

                if(DIR_VLD= '1' and (DIR=dir_dpram1 or DIR=dir_dpram2))then
                    state<=s1;
                end if;

            WHEN s1 =>

                if (DATO_VLD= '1' and dirActual=dirPrev)then
                    state<=s2;
                elsif(DATO_VLD= '1' and dirActual/=dirPrev) then
                    state<=s3;
                end if;

                WHEN s2 =>

                    CCounter<='1';
                    state<=s4;

                WHEN s3 =>

                    clearCounter<='1';
                    state<=s4;

                WHEN s4 =>

                    if(DIR=dir_dpram1)then
                        we_dp1<='1';
                    elsif (DIR=dir_dpram2) then
                        we_dp2<='1';
                    end if;
                    dirPrev<=DIR;
                    CCounter<='0';
                    clearCounter<='0';
                    state<=s0;

                end case;
            end if;
        end process;
    end RTL;
```

B) Código VHDL del testbench entidad *cnt_dpram_tb*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY cnt_dpram_tb IS
END cnt_dpram_tb;

ARCHITECTURE behavior OF cnt_dpram_tb IS
    COMPONENT cnt_dpram
    PORT(
        CLK : IN std_logic;
        RST : IN std_logic;
        DIR : IN std_logic_vector(7 downto 0);
        DIR_VLD : IN std_logic;
        DATO : IN std_logic_vector(7 downto 0);
        DATO_VLD : IN std_logic;
        ADDRESS : OUT std_logic_vector(7 downto 0);
        DATA : OUT std_logic_vector(7 downto 0);
        WE_DP1 : OUT std_logic;
        WE_DP2 : OUT std_logic
    );
    END COMPONENT;

    --Inputs
    signal CLK : std_logic := '0';
    signal RST : std_logic := '1';
    signal DIR : std_logic_vector(7 downto 0) := (others => '0');
    signal DIR_VLD : std_logic := '0';
    signal DATO : std_logic_vector(7 downto 0) := (others => '0');
    signal DATO_VLD : std_logic := '0';
    --Outputs
    signal ADDRESS : std_logic_vector(7 downto 0);
    signal DATA : std_logic_vector(7 downto 0);
    signal WE_DP1 : std_logic;
    signal WE_DP2 : std_logic;

    -- Clock period definitions
    constant CLK_period : time := 10 ns;
BEGIN
    uut: cnt_dpram PORT MAP (
        CLK => CLK,
        RST => RST,
        DIR => DIR,
        DIR_VLD => DIR_VLD,
        DATO => DATO,
        DATO_VLD => DATO_VLD,
        ADDRESS => ADDRESS,
        DATA => DATA,
        WE_DP1 => WE_DP1,
        WE_DP2 => WE_DP2
    );
    CLK_process :process
    begin
        CLK <= '0';
        wait for CLK_period/2;
        CLK <= '1';
        wait for CLK_period/2;
    end process;

    RST_process :process
    begin
        wait for 20 ns;
        RST <= '0';
    end process;
```

```
DATO_process :process
    begin
        if(rst='1')then
            DATO<=(others=>'0');
            wait for 500 ns;
        end if;

        DATO <= x"55";
        DATO_VLD<='1';
        wait for 10 ns;
        DATO_VLD<='0';
        wait for 490 ns;
        DATO_VLD<='1';
        DATO <= x"69";
        wait for 10 ns;
        DATO_VLD<='0';
        wait for 490 ns;
        DATO_VLD<='1';
        DATO <= x"85";
        wait for 10 ns;
        DATO_VLD<='0';
        wait for 490 ns;
        DATO_VLD<='1';
        DATO <= x"42";
        wait for 10 ns;
        DATO_VLD<='0';
        wait for 490 ns;
        DATO_VLD<='1';
        DATO <= x"65";
        wait for 10 ns;
        DATO_VLD<='0';
        wait for 490 ns;
        DATO_VLD<='1';
        DATO <= x"00";
        wait for 10 ns;
        wait;
    end process;

    begin
        if(rst='1')then
            DIR<=(others=>'0');
            wait for 250 ns;
        end if;

        DIR <= x"a2";
        DIR_VLD<='1';
        wait for 10 ns;
        DIR_VLD<='0';
        wait for 490 ns;
        DIR_VLD<='1';
        DIR <= x"a1";
        wait for 10 ns;
        DIR_VLD<='0';
        wait for 490 ns;
        DIR_VLD<='1';
        DIR <= x"a2";
        wait for 10 ns;
        DIR_VLD<='0';
        wait for 490 ns;
        DIR_VLD<='1';
        DIR <= x"a1";
        wait for 10 ns;
        DIR_VLD<='0';
        wait for 490 ns;
        DIR_VLD<='1';
        DIR <= x"a1";
        wait for 10 ns;
        DIR_VLD<='0';
        wait for 490 ns;
        DIR_VLD<='1';
        DIR <= x"a1";
        wait for 10 ns;
        wait;
```

5.1.2- Razonamiento de por qué se ha adoptado la solución presentada.

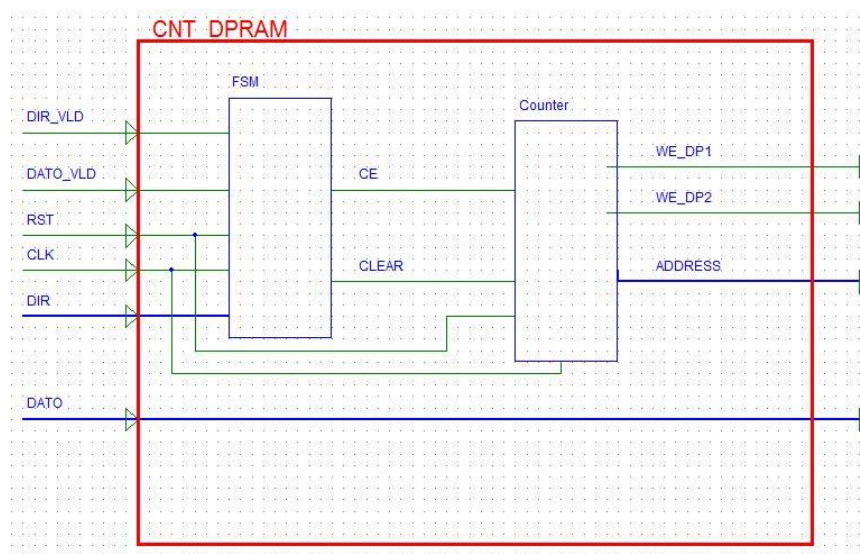
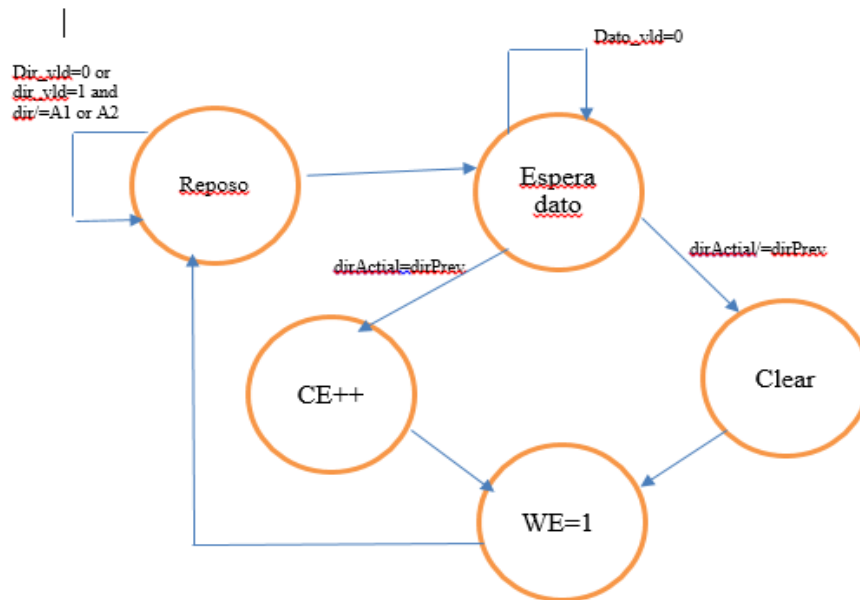
Este módulo es el controlador de las memorias dual port. Su misión es indicar que dirección de la memoria debe ser accedida dada el módulo de memoria a donde se quiera acceder.

Por ejemplo; si se desea acceder el módulo de memoria A1h repetidas veces, este módulo se encarga de escribir indicar posiciones distintas de escritura dentro del mismo módulo de memoria A1h

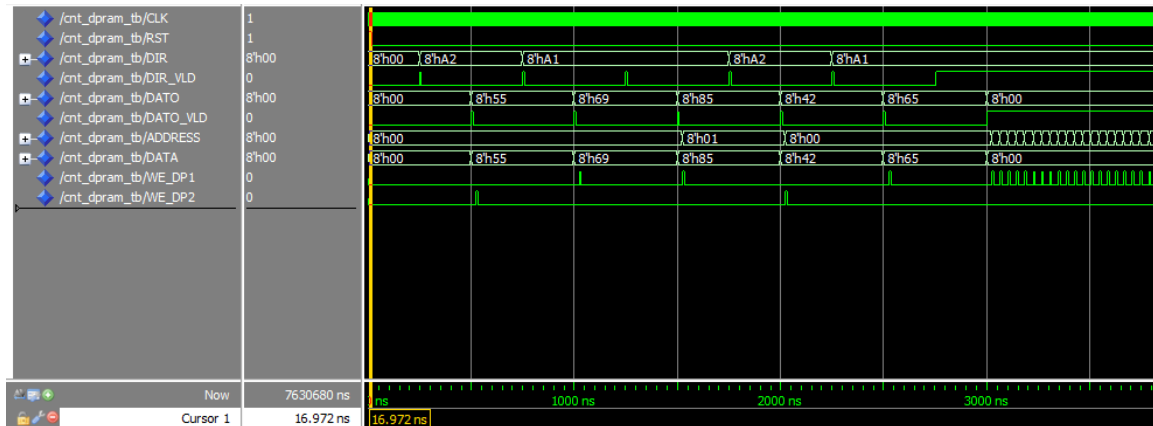
El modulo está compuesto por un contador y un una máquina de estados.

El funcionamiento de la máquina de estados regula el funcionamiento general del sistema.

La máquina de estados se encarga de estudiar cuando se ha recibido una nueva dirección y comprueba si es una dirección para el mismo modulo o para el modulo asignado la ejecución anterior.



5.1.3.- Pantallazo de simulación donde se refleje que el mencionado circuito funciona correctamente. Para ello en el banco de pruebas deberá incluir los estímulos necesarios para comprobar que el módulo funciona correctamente.



Esta grafica muestra el funcionamiento del módulo.

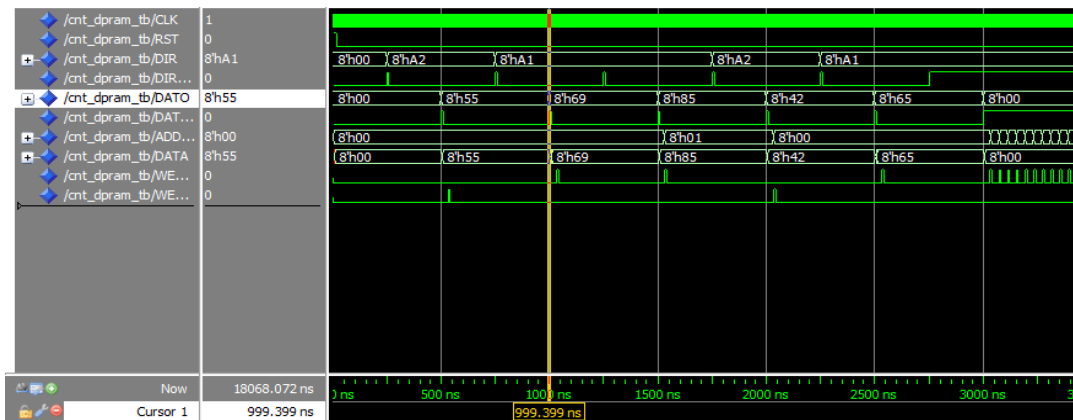
El proceso se inicializa con DIR=A2 y una vez que se recibe DATO_VLD la salida ADDRESS escribe un 0 ya que es la primera vez que se escribe en la memoria A2. El siguiente paso de DIR es A1, y se aprecia como ADDRESS mantiene el valor 0 ya que también es la primera vez que se accede la memoria A1. El siguiente ciclo vuelve a ser DIR=A1, como esta es la segunda referencia consecutiva a esta memoria ADDRESS es ahora 01h.

5.2.1.- Tabla donde se incluyan los recursos utilizados para la implementación.

Device Utilization Summary					[-]
Slice Logic Utilization	Used	Available	Utilization	Note(s)	
Number of Slice Registers	31	54,576	1%		
Number used as Flip Flops	31				
Number used as Latches	0				
Number used as Latch-thrus	0				
Number used as AND/OR logics	0				
Number of Slice LUTs	21	27,288	1%		
Number used as logic	21	27,288	1%		
Number using O6 output only	15				
Number using O5 output only	0				
Number using O5 and O6	6				
Number used as ROM	0				
Number used as Memory	0	6,408	0%		
Number of occupied Slices	10	6,822	1%		
Number of MUXCYs used	0	13,644	0%		
Number of LUT Flip Flop pairs used	34				
Number with an unused Flip Flop	4	34	11%		
Number with an unused LUT	13	34	38%		

Number of fully used LUT-FF pairs	17	34	50%	
Number of unique control sets	4			
Number of slice register sites lost to control set restrictions	1	54,576	1%	
Number of bonded IOBs	38	218	17%	
Number of RAMB16BWERs	0	116	0%	
Number of RAMB8BWERs	0	232	0%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%	
Number of BUFG/BUFGMUXs	1	16	6%	
Number used as BUFGs	1			
Number used as BUFGMUX	0			
Number of DCM/DCM_CLKGENs	0	8	0%	
Number of ILOGIC2/ISERDES2s	0	376	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	376	0%	
Number of OLOGIC2/OSERDES2s	0	376	0%	
Number of BSCANs	0	4	0%	
Number of BUFHs	0	256	0%	
Number of BUFPLLs	0	8	0%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	58	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	0	4	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	3.13			

5.2.2.- Pantallazo de simulación donde se refleje que el módulo de representación funciona correctamente.



El comportamiento de la simulación temporal es similar a la simulación funcional. El funcionamiento sigue siendo correcto.

6. GEN_DIR

6.1.1.- Código VHDL del módulo y del banco de pruebas utilizado en la simulación.

A) Código VHDL de la entidad *gen_dir*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity gen_dir is
generic ( N : integer := 68 );
port (
CLK    : in std_logic;
RST    : in std_logic;
DIR    : in std_logic_vector(7 downto 0);
DIR_VLD : in std_logic;
DATO    : in std_logic_vector(7 downto 0);
DATO_VLD : in std_logic;
ADDR_OUT : out std_logic_vector(7 downto 0);
DATO_OK : out std_logic);
end gen_dir;
architecture rtl of gen_dir is
constant dir_freq : std_logic_vector( 7 downto 0) := x"F0";

type state_type is (
s0, --HOLDING STATE
s1 --WAIT DATA STATE
);
signal state: state_type; --current and next state declaration.
signal valor_freq: std_logic_vector( 7 downto 0);
signal cnt: std_logic_vector( 15 downto 0);
signal valor_freq_comm: std_logic_vector( 15 downto 0);
signal CE: std_logic;
signal prescalerCounter: unsigned(6 downto 0); --7 bits to cover generic 68
signal dato_ok_trigger: std_logic;
begin
prescaler_proc:process (CLK,RST)
begin
if (RST='1') then
CE <= '0';
prescalerCounter<=(others=>'0');
elsif (CLK'event and CLK='1')then
CE <= '0';
prescalerCounter<=prescalerCounter+1;
if (prescalerCounter = N)then
CE <= '1';
prescalerCounter<=(others=>'0');
end if;
end if;
end process;
fsm_proc:process(clk, rst,valor_freq,DIR_VLD,DIR,DATO_VLD)
begin
if(rst = '1') then
state <= s0;
valor_freq<=(others=>'0');
elsif (clk'event and clk = '1') then
case state is
WHEN s0 =>
if(DIR_VLD= '1' and DIR=dir_freq)then
state<=s1;
end if;
WHEN s1 =>
if (DATO_VLD= '1')then
valor_freq<=DATO;
state<=s0;
end if;
end case;
end if;
end process;
```

```
valor_freq_comm<=std_logic_vector(unsigned(valor_freq) + unsigned(cnt));

registro:process (clk,rst,valor_freq_comm,CE)
begin
if (rst='1') then
cnt <= (others=>'0');
elsif (clk'event and clk='1')then
if (CE = '1')then
cnt<=valor_freq_comm;
end if;
end if;
end process;

ADDR_OUT<=cnt(15 downto 8);
dato_ok_trigger<='1' when (cnt /= x"0000") else ('0');
data_ok_proc:process (clk,rst,prescalerCounter)
begin
if (rst='1') then
DATO_OK<='0';
elsif (clk'event and clk='1')then
if (prescalerCounter = 1)then
DATO_OK<='1';
else
DATO_OK<='0';
end if;
end if;
end process;
end rtl;
```

B) Código VHDL del testbench de la entidad *gen_dir_tb*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY gen_dir_tb IS
END gen_dir_tb;
ARCHITECTURE behavior OF gen_dir_tb IS
    COMPONENT gen_dir
        PORT(
            CLK : IN std_logic;
            RST : IN std_logic;
            DIR : IN std_logic_vector(7 downto 0);
            DIR_VLD : IN std_logic;
            DATO : IN std_logic_vector(7 downto 0);
            DATO_VLD : IN std_logic;
            ADDR_OUT : OUT std_logic_vector(7 downto 0);
            DATO_OK : OUT std_logic
        );
    END COMPONENT;
--Inputs
signal CLK : std_logic := '0';
signal RST : std_logic := '1';
signal DIR : std_logic_vector(7 downto 0) := (others => '0');
signal DIR_VLD : std_logic := '0';
signal DATO : std_logic_vector(7 downto 0) := (others => '0');
signal DATO_VLD : std_logic := '0';
--Outputs
signal ADDR_OUT : std_logic_vector(7 downto 0);
signal DATO_OK : std_logic;
-- Clock period definitions
constant CLK_period : time := 10 ns;

BEGIN
-- Instantiate the Unit Under Test (UUT)
ut: gen_dir PORT MAP (
    CLK => CLK,
    RST => RST,
    DIR => DIR,
    DIR_VLD => DIR_VLD,
    DATO => DATO,
    DATO_VLD => DATO_VLD,
    ADDR_OUT => ADDR_OUT,
    DATO_OK => DATO_OK
);
CLK_process :process
begin
CLK <= '0';
wait for CLK_period/2;
CLK <= '1';
wait for CLK_period/2;
end process;
RST_process :process
begin
wait for 10 ns;
RST <= '0';
end process;
```

```
DATO_process :process
begin
if (rst='1')then
    DATO<=(others=>'0');
wait for 500 ns;
end if;
DATO <= x"55";
DATO_VLD<='1';
wait for 10 ns;
DATO_VLD<='0';
wait for 490 ns;
DATO_VLD<='1';
DATO <= x"69";
wait for 10 ns;
DATO_VLD<='0';
wait for 490 ns;
DATO_VLD<='1';
DATO <= x"85";
wait for 10 ns;
DATO_VLD<='0';
wait for 490 ns;
DATO_VLD<='1';
DATO <= x"42";
wait for 10 ns;
DATO_VLD<='0';
wait for 490 ns;
DATO_VLD<='1';
DATO <= x"65";
wait for 10 ns;
DATO_VLD<='0';
wait for 490 ns;
DATO_VLD<='1';
DATO <= x"00";
wait for 10 ns;
DATO_VLD<='0';
wait for 490 ns;
DATO_VLD<='0';
DATO <= x"00";
wait for 10 ns;
wait;
end process;
```

```
DIR_VLD_process :process
begin
if (rst='1')then
    DIR<=(others=>'0');
wait for 250 ns;
end if;

DIR <= x"a2";
DIR_VLD<='1';
wait for 10 ns;
DIR_VLD<='0';
wait for 490 ns;
DIR_VLD<='1';
DIR <= x"a1";
wait for 10 ns;
DIR_VLD<='0';
wait for 490 ns;
DIR_VLD<='1';
DIR <= x"F0";
wait for 10 ns;
DIR_VLD<='0';
wait for 490 ns;
DIR_VLD<='1';
DIR <= x"a2";
wait for 10 ns;
DIR_VLD<='0';
wait for 490 ns;
DIR_VLD<='1';
DIR <= x"a1";
wait for 10 ns;
DIR_VLD<='0';
wait for 490 ns;
DIR_VLD<='1';
DIR <= x"F0";
wait for 10 ns;
DIR_VLD<='0';
wait for 490 ns;
DIR_VLD<='0';
DIR <= x"00";
wait for 10 ns;
wait;
end process;

END;
```

6.1.2- Razonamiento de por qué se ha adoptado la solución presentada.

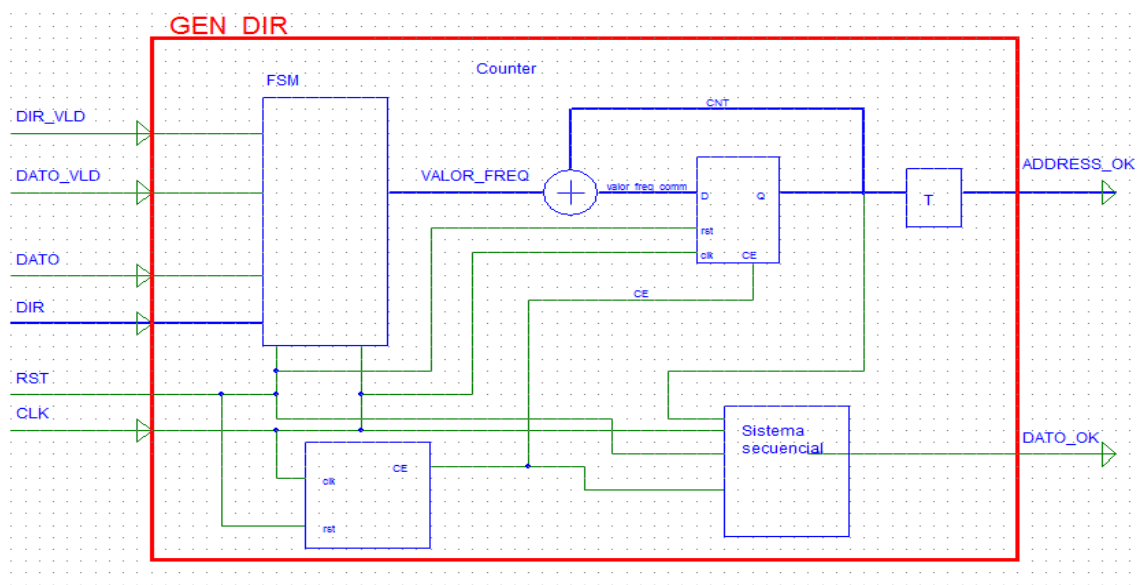
Siguiendo el esquema proporcionado, se han creado los siguientes componentes;

Un prescaler con un factor de división de 68 pulsos de CLK.

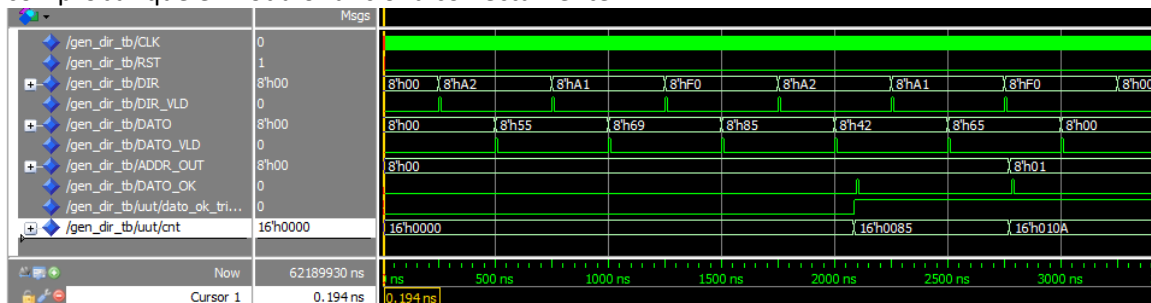
Una máquina de estados que espera a una dirección válida y un dato válido. Una vez recibido ambos valores válidos, comprueba a su vez que la DIR introducida corresponde con la dirección F0h. Si es así el valor del puerto DATO de entrada se suma al valor que sale del biestable.

El biestable mantiene el valor del DATO durante un ciclo del prescaler y retorna el valor al sumador para sumarlo con un nuevo valor de entrada.

Por último se ha diseñado un sistema secuencial en el cual, cuando el contador del prescaler alcanza el valor 1 y la salida de cnt es distinto al inicio ("0000"h) se activa la salida DATO_OK. Se ha tomado el inicio en 0h de cnt porque es un valor que tras ser sumado repetidas veces dos cualesquiera nunca originara el valor 0h.




6.1.3.- Pantallazo de simulación donde se refleje que el mencionado circuito funciona correctamente. Para ello en el banco de pruebas deberá incluir los estímulos necesarios para comprobar que el módulo funciona correctamente.



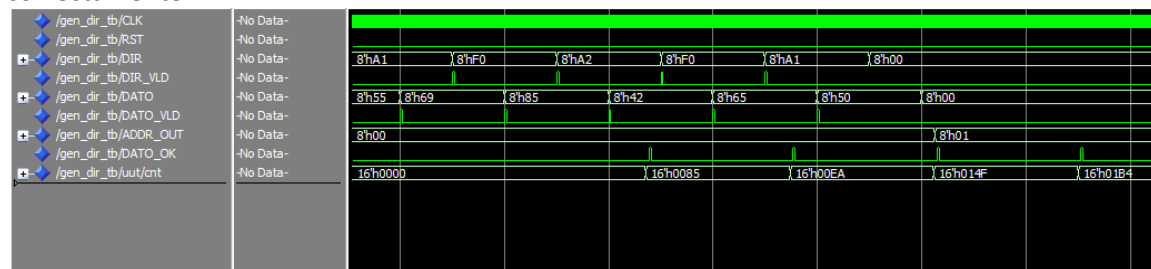
En la gráfica anterior cuando llega una dirección en DIR igual a F0h y se detecta un dato válido (DATO_VLD=1) y la salida para el primer caso de ADDR_OUT es igual a 00h.

6.2.1.- Tabla donde se incluyan los recursos utilizados para la implementación.

Device Utilization Summary				
Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	34	54,576	1%	
Number used as Flip Flops	34			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	32	27,288	1%	
Number used as logic	31	27,288	1%	
Number using O6 output only	22			
Number using O5 output only	7			
Number using O5 and O6	2			
Number used as ROM	0			
Number used as Memory	0	6,408	0%	
Number used exclusively as route-thrus	1			
Number with same-slice register load	0			
Number with same-slice carry load	1			
Number with other load	0			
Number of occupied Slices	14	6,822	1%	
Number of MUXCYs used	16	13,644	1%	
Number of LUT Flip Flop pairs used	40			
Number with an unused Flip Flop	8	40	20%	
Number with an unused LUT	8	40	20%	
Number of fully used LUT-FF pairs	24	40	60%	
Number of unique control sets	3			
Number of slice register sites lost to control set restrictions	6	54,576	1%	
Number of bonded IOBs	29	218	13%	
Number of RAMB16BWERs	0	116	0%	
Number of RAMB8BWERs	0	232	0%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%	
Number of BUFG/BUFGMUXs	1	16	6%	
Number used as BUFGs	1			
Number used as BUFGMUX	0			
Number of DCM/DCM_CLKGENs	0	8	0%	
Number of ILOGIC2/ISERDES2s	0	376	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	376	0%	
Number of OLOGIC2/OSERDES2s	0	376	0%	
Number of BSCANs	0	4	0%	
Number of BUFHs	0	256	0%	
Number of BUFPLLs	0	8	0%	

Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	58	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	0	4	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	2.97			

6.2.2.- Pantallazo de simulación temporal donde se refleje que el módulo de representación funciona correctamente.



Analizando la simulación temporal se ve que con la primera lectura de DRI=F0 se obtiene el dato 85h, los bits de mayor peso pasan a ADDR_OUT, es decir el valor será de 0h y se activa el DATO_OK. El segundo valor de lectura para F= es 65h sumando ambos se obtiene EAh. En ADDR_OUT de nuevo solo pasan los de mayor peso luego sigue siendo 0h. Una vez pasado el periodo del prescaler, como no hay una entrada valida nueva se mantiene el valor anterior (65h) que sumado al valor anterior EAh resulta en 014F. Este nuevo valor provoca en la salida ADDR_OUT el valor de 01h ya que solo coge los de mayor peso del bus CNT.

7. GEN_FUNCIONES

7.1.1.- Código VHDL del banco de pruebas utilizado para su simulación. La entidad *gen_funciones* se proporciona con el enunciado.

A) Código VHDL de la entidad *gen_funciones_tb*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY gen_funciones_tb IS
END gen_funciones_tb;

ARCHITECTURE behavior OF gen_funciones_tb IS

    COMPONENT gen_funciones
    PORT(
        RELOJ : IN std_logic;
        RST : IN std_logic;
        ASTRB : IN std_logic;
        DSTRB : IN std_logic;
        DATA : INOUT std_logic_vector(7 downto 0);
        PWRITE : IN std_logic;
        PWAIT : OUT std_logic;
        SYNC : OUT std_logic;
        SCLK : OUT std_logic;
        D1 : OUT std_logic;
        D2 : OUT std_logic
    );
    END COMPONENT;

    COMPONENT epp_device1
        port (
            DATA : inout std_logic_vector(7 downto 0);
            PWRITE : out std_logic;
            DSTRB : out std_logic;
            ASTRB : out std_logic;
            PWAIT : in std_logic);
    END COMPONENT;

    COMPONENT DAC121S101
        port (
            VOUT : out real range 0.0 to 3.5;
            SYNC : in std_logic;
            SCLK : in std_logic;
            DIN : in std_logic);
    end COMPONENT;

    --Inputs
    signal RELOJ : std_logic := '0';
    signal RST : std_logic := '1';
    signal ASTRB : std_logic := '0';
    signal DSTRB : std_logic := '0';
    signal PWRITE : std_logic := '0';

    --BiDirs
    signal DATA : std_logic_vector(7 downto 0):=x"00";
    --Outputs
    signal VOUT1 : real range 0.0 to 3.5;
    signal VOUT2 : real range 0.0 to 3.5;
    signal PWAIT : std_logic;
    signal SYNC : std_logic;
    signal SCLK : std_logic;
    signal D1 : std_logic;
    signal D2 : std_logic;

    -- Clock period definitions
    constant RELOJ_period : time := 10 ns;
```

```
BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: gen_funciones PORT MAP (
    RELOJ => RELOJ,
    RST => RST,
    ASTRB => ASTRB,
    DSTRB => DSTRB,
    DATA => DATA,
    PWRITE => PWRITE,
    PWAIT => PWAIT,
    SYNC => SYNC,
    SCLK => SCLK,
    D1 => D1,
    D2 => D2
);
eppDevice:epp_device1
    port map(
        DATA=>DATA,
        PWRITE=>PWRITE,
        DSTRB=>DSTRB,
        ASTRB=>ASTRB,
        PWAIT=>PWAIT
    );

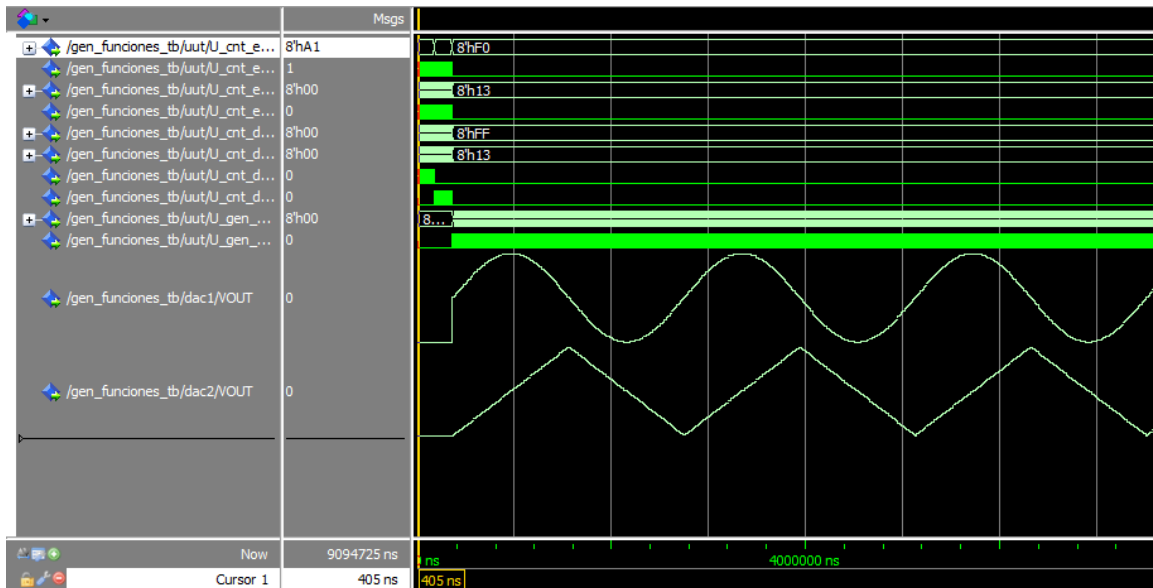
dac1:DAC121S101
    port map(
        VOUT => VOUT1,
        SYNC => SYNC,
        SCLK => SCLK,
        DIN =>D1);

dac2:DAC121S101
    port map(
        VOUT=> VOUT2,
        SYNC => SYNC,
        SCLK => SCLK,
        DIN =>D2);

-- Clock process definitions
RELOJ_process :process
begin
    RELOJ <= '0';
    wait for RELOJ_period/2;
    RELOJ <= '1';
    wait for RELOJ_period/2;
end process;

RST_process :process
begin
    wait for 100 ns;
    RST <= '0';
end process;
END;
```

7.1.2.- Pantallazo de simulación donde se refleje que el mencionado circuito funciona correctamente. Para ello en el banco de pruebas deberá incluir los estímulos necesarios para comprobar que el módulo funciona correctamente.



La primera señal muestra la entrada de las direcciones que dirigen el funcionamiento del sistema y se ve como empieza en A1. En esta posición guarda un número de datos, y posteriormente hace lo mismo para A2. Por ultimo con F0 comienzan a leerse los datos e interpretarse como señales sinodales.

7.2.1.- Tabla donde se incluyan los recursos utilizados para la implementación

Device Utilization Summary				[-]
Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	94	54,576	1%	
Number used as Flip Flops	94			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	68	27,288	1%	
Number used as logic	67	27,288	1%	
Number using O6 output only	46			
Number using O5 output only	7			
Number using O5 and O6	14			
Number used as ROM	0			
Number used as Memory	0	6,408	0%	
Number used exclusively as route-thrus	1			
Number with same-slice register load	0			
Number with same-slice carry load	1			
Number with other load	0			

Number of occupied Slices	35	6,822	1%	
Number of MUXCYs used	16	13,644	1%	
Number of LUT Flip Flop pairs used	105			
Number with an unused Flip Flop	16	105	15%	
Number with an unused LUT	37	105	35%	
Number of fully used LUT-FF pairs	52	105	49%	
Number of unique control sets	9			
Number of slice register sites lost to control set restrictions	10	54,576	1%	
Number of bonded IOBs	18	218	8%	
Number of LOCed IOBs	18	18	100%	
IOB Flip Flops	1			
Number of RAMB16BWERs	0	116	0%	
Number of RAMB8BWERs	2	232	1%	
Number of BUFIO2/BUFIO2_2CLKs	1	32	3%	
Number used as BUFIO2s	1			
Number used as BUFIO2_2CLKs	0			
Number of BUFIO2FB/BUFIO2FB_2CLKs	1	32	3%	
Number used as BUFIO2FBs	1			
Number used as BUFIO2FB_2CLKs	0			
Number of BUFG/BUFGMUXs	1	16	6%	
Number used as BUFGs	1			
Number used as BUFGMUX	0			
Number of DCM/DCM_CLKGENs	1	8	12%	
Number used as DCMs	1			
Number used as DCM_CLKGENs	0			
Number of ILOGIC2/ISERDES2s	0	376	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	376	0%	
Number of OLOGIC2/OSERDES2s	1	376	1%	
Number used as OLOGIC2s	1			
Number used as OSERDES2s	0			
Number of BSCANs	0	4	0%	
Number of BUFHs	0	256	0%	
Number of BUFPLLs	0	8	0%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	58	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	0	4	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	3.52			

7.2.2.- Pantallazo de simulación donde se refleje que el sistema funciona correctamente.